1989

Computer implementation and simulation of some neural networks used in pattern recognition and classification.

Khaidar, Mohamed H.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

COMPUTER IMPLEMENTATION AND SIMULATION OF SOME NEURAL NETWORKS USED IN PATTERN RECOGNITION AND CLASSIFICATION

by

Mohamed H. Khaidar

March 1989

Thesis Advisor Tri T. Ha

REPORT DOCUMENTATION PAGE

Report Security Classification Unclassified		1b Restrictive Markings	
Security Classification Authority		3 Distribution Availability of Report	
Declassification Downgrading Schedule		Approved for public release; distribution is unlimited.	
Performing Organization Report Number(s)		5 Monitoring Organization Report Number(s)	
Name of Performing Organization val Postgraduate School	6b Office Symbol *(if applicable)* 32	7a Name of Monitoring Organization Naval Postgraduate School	
Address *(city, state, and ZIP code)* onterey, CA 93943-5000		7b Address *(city, state, and ZIP code)* Monterey, CA 93943-5000	
Name of Funding Sponsoring Organization	8b Office Symbol *(if applicable)*	9 Procurement Instrument Identification Number	
Address *(city, state, and ZIP code)*		10 Source of Funding Numbers	
		Program Element No / Project No / Task No / Work Unit Accession No	

Title *(include security classification)* COMPUTER IMPLEMENTATION AND SIMULATION OF SOME NEURAL NETORKS USED IN PATTERN RECOGNITION AND CLASSIFICATION

Personal Author(s) Mohamed H. Khaidar

Type of Report	13b Time Covered	14 Date of Report *(year, month, day)*	15 Page Count
ster's Thesis	From To	March 1989	131

Supplementary Notation The views expressed in this thesis are those of the author and do not reflect the official policy or poon of the Department of Defense or the U.S. Government.

Cosati Codes			18 Subject Terms *(continue on reverse if necessary and identify by block number)*
d	Group	Subgroup	Neural network, Hopfield net, Hamming net, Carpenter / Grossberg net, pattern

Abstract *(continue on reverse if necessary and identify by block number)*

Searchers and scientists have been studying neural networks for many years hoping to achieve human-like performance he fields of speech and pattern recognition and classification. In this study, we are first going to make an introduction to field of artificial neural networks, then we are going to describe some of the neural nets used in the pattern recognition d classification. A computer simulation program from an algorithmic approach for each one of these networks will be nstructed and used to implement the operation of the net. Its ability will be demonstrated in differentiating between different tterns and even correcting a noisy pattern and recognizing it. The Hopfield network, the Hamming network and the Carnter / Grossberg network will be individually utilized in developing an algorithm for pattern recognition and classification. The maximum-likelihood sequence estimation function will be mapped onto a neural network structure. The application his structure computations for data detection in digital communications receivers will be described. A computer simulation ogram will be constructed and used to show that neural networks offer attractive implementation alternatives for MLSE.

Distribution Availability of Abstract nclassified unlimited ☐ same as report ☐ DTIC users	21 Abstract Security Classification Unclassified	
Name of Responsible Individual T. Ha	22b Telephone *(include Area code)* (408) 384-2991	22c Office Symbol 62Ha

FORM 1473,84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

security classification of this page
Unclassified

Approved for public release; distribution is unlimited.

Computer Implementation and Simulation of Some Neural Networks Used in Pattern Recognition and Classification

by

Mohamed H. Khaidar
LTJG, Royal Moroccan Navy
B.S., Royal Naval Academy, 1986

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 1989

ABSTRACT

Searchers and scientists have been studying neural networks for many years hoping to achieve human-like performance in the fields of speech and pattern recognition and classification. In this study, we are first going to make an introduction to the field of artificial neural networks, then we are going to describe some of the neural nets used in the pattern recognition and classification. A computer simulation program from an algorithmic approach for each one of these networks will be constructed and used to implement the operation of the net. Its ability will be demonstrated in differentiating between different patterns and even correcting a noisy pattern and recognizing it. The Hopfield network, the Hamming network and the Carpenter / Grossberg network will be individually utilized in developing an algorithm for pattern recognition and classification.

The maximum-likelihood sequence estimation function will be mapped onto a neural network structure. The application of this structure computations for data detection in digital communications receivers will be described. A computer simulation program will be constructed and used to show that neural networks offer attractive implementation alternatives for MLSE.

TABLE OF CONTENTS

I. INTODUCTION 1
A. WHAT IS A NEURAL NETWORK : 1
B. NEURAL NETWORKS IN PATTERN RECOGNITION AND CLASSIFICATION : 3
C. NEURAL NETWORKS AS MLSE RECEIVERS OF BINARY SIGNALS IN GAUSSIAN NOISE : 5

II. THE HOPFIELD NETWORK 6
A. GENERALITIES: 6
B. OPERATION OF THE HOPFIELD NETWORK: 6
C. IMPLEMENTATION OF THE HOPFIELD NETWORK: 9
D. SIMULATION OF THE HOPFIELD NETWORK: 12

III. THE HAMMING NETWORK 19
A. GENERALITIES : 19
B. OPERATION OF THE HAMMING NETWORK : 20
C. IMPLEMENTATION OF THE HAMMING NET : 24
D. SIMULATION OF THE HAMMING NETWORK: 27

IV. THE CARPENTER / GROSSBERG NET 49
A. GENERALITIES : 49
B. IMPLEMENTATION OF THE CARPENTER / GROSSBERG NET : 49
C. SIMULATION OF THE CARPENTER / GROSSBERG NET : 53

V. NEURAL NETWORK AS A BINARY MAXIMUM-LIKELIHOOD SEQUENCE ESTIMATOR 60
A. GENERALITIES : 60
B. MAXIMUM-LIKELIHOOD SEQUENCE ESTIMATION : 60
C. NEURAL NETWORK : 63
D. MAPPING OF MLSE ONTO A NEURAL NETWORK : 66
E. NEURAL NETWORK MAXIMUM-LIKELIHOOD RECEIVER : 68

F. SIMULATIONS AND RESULTS : 70

VI. CONCLUSION 75
A. SUMMARY OF RESULTS : 75
B. NEURAL NETWORK TASKS : 77
C. CONCLUSIONS : 78

APPENDIX A. PROGRAMING THE HOPFIELD NET WHEN USED AS A CLASSIFIER : 80

APPENDIX B. PROGRAMING THE HAMMING NET WHEN USED AS AN OPTIMUM CLASSIFIER : 88

APPENDIX C. ART AND OPERATION OF THE CARPENTER / GROSSBERG NET : 95

APPENDIX D. PROGRAMING THE CARPENTER / GROSSBERG NET ... 98

APPENDIX E. THE PARAMETERS FOR THE MLSE NEURAL NETWORK 106

APPENDIX F. PROGRAMMING THE MLSE NEURAL NETWORK 108

LIST OF REFERENCES 120

INITIAL DISTRIBUTION LIST 122

LIST OF TABLES

Table 1. SIMULATIONS RESULTS FOR MLSE NEURAL NETWORK (STATIONARY CHANNEL) 73
Table 2. SIMULATION RESULTS FOR MLSE NEURAL NETWORK (TIME-VARYING CHANNEL) 74

LIST OF FIGURES

Figure 1. Neural Network and a Nodal Preprocessing Element [Ref. 1] 1
Figure 2. Biological Neurons and a Small Biological Neural Network [Ref. 1] 2
Figure 3. A taxonomy of Neural Network Classification and Clustering Models [Ref. 2] 4
Figure 4. The hard-limiting function used in the Hopfield network 7
Figure 5. The Hopfield net used as a content-addressble memory [Ref. 2] 9
Figure 6. The Eight stored patterns 11
Figure 7. Hopfielf net response to the first input pattern 13
Figure 8. Hopfield net response to the second input pattern 14
Figure 9. Classification response of the Hopfield net to the second input pattern . 15
Figure 10. Hopfield net response to the third input pattern 16
Figure 11. Hopfield net response to the perfect input pattern 17
Figure 12. Feed-forward neural net used to calculate M weighted sums from the N elements of the input pattern [Ref. 4] 21
Figure 13. The iterative neural net called "maxnet" that picks the maximum of M inputs [Ref. 4] 22
Figure 14. The complete neural network classifier referred to as The Hamming net [Ref. 2] 24
Figure 15. The first four stored patterns 26
Figure 16. The rest of the 10 stored patterns 27
Figure 17. Response of the Hamming net to the first input pattern 28
Figure 18. The output of the Hamming net at t = 1 for digit "3" 29
Figure 19. The output of the Hamming net at t = 2 for digit "3" 30
Figure 20. The output of the Hamming net at t = 3 for digit "3" 31
Figure 21. The output of the Hamming net at t = 4 for digit "3" 32
Figure 22. The output of the Hamming net at t = 5 for digit "3" 33
Figure 23. The output of the Hamming net at t = 6 for digit "3" 34
Figure 24. The output of the Hamming net at t = 7 for digit "3" 35
Figure 25. The output of the Hamming net at t = 8 for digit "3" 36
Figure 26. Response of the Hamming net to the second input pattern 39
Figure 27. The output of the Hamming net at t = 1 for digit "9" 40

Figure 28. The output of the Hamming net at t = 2 for digit "9" 41
Figure 29. The output of the Hamming net at t = 3 for digit "9" 42
Figure 30. The output of the Hamming net at t = 4 for digit "9" 43
Figure 31. The output of the Hamming net at t = 5 for digit "9" 44
Figure 32. The output of the Hamming net at t = 6 for digit "9" 45
Figure 33. The output of the Hamming net at t = 7 for digit "9" 46
Figure 34. Response of the Hamming net to the perfect input pattern 47
Figure 35. The major components of the Carpenter / Grossberg classification net [Ref. 2] 51
Figure 36. Adaptive Maximum-Likelihood Receiver [Ref. 3] 61
Figure 37. Hopfield Neural Network [Ref. 3] 64
Figure 38. Matrix of synaptic connections [Ref. 3] 67
Figure 39. MLSE neural network [Ref. 3] 67
Figure 40. Neural Network Based Maximum-Likelihood Receiver [Ref. 3] 69
Figure 41. Seven Tasks that Neural Networks Can Perform [Ref. 1] 77
Figure 42. The ART net search for a correct F_2 code. [Ref. 12] 95

I. INTRODUCTION

A. WHAT IS A NEURAL NETWORK :

A neural network is a highly parallel network with many interconnections between analog computational elements or nodes. In other words, a neural net is a system composed of many simple processing elements operating in parallel whose function is determined by network structure, connection strengths and the processing performed at computing elements or nodes. These nodes offer one possible solution to the problem of obtaining the massive parallelism and computational requirements that are presumed to be required for such problems as pattern recognition and classification that we are going to discuss in this study. Artificial neural nets are of interest primarily because they may be able to emulate the speed and performance of real biological neural nets using many simple slow computational elements operating in parallel to obtain high computation rates. Figure 1 illustrates this definition.

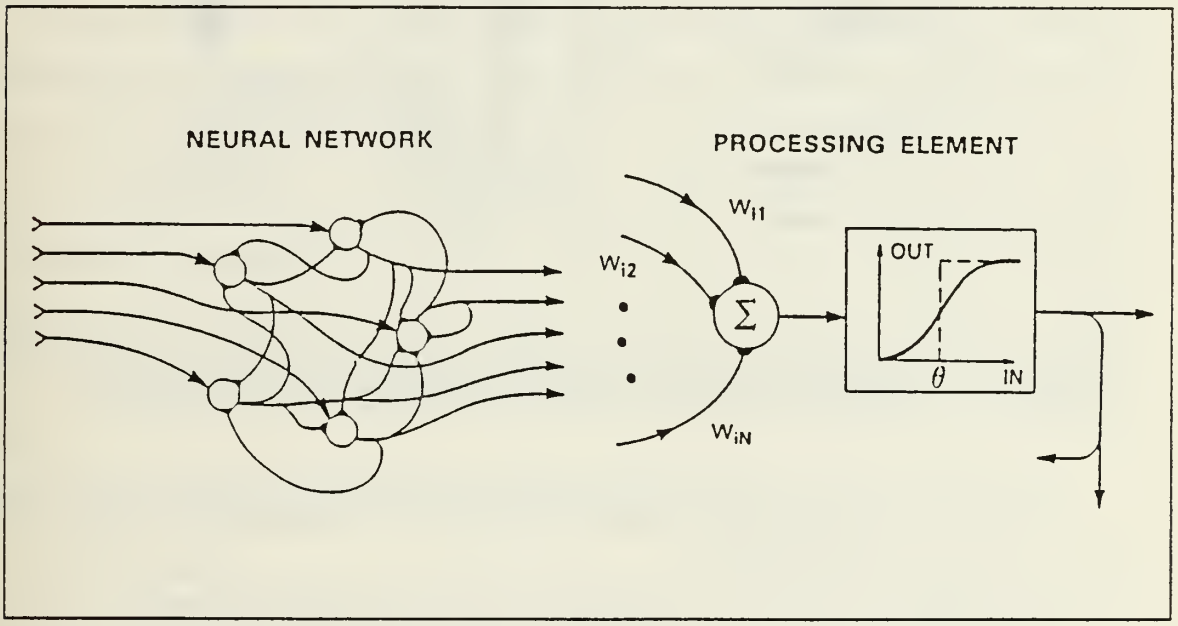


Figure 1. Neural Network and a Nodal Preprocessing Element [Ref. 1]

A small interconnected neural network is presented on the left side of this figure and one simple type of processing element or node is presented on the right side [Ref. 1]. This particular node forms the sum of N weighted inputs presented on N input links and

passes the result through a nonlinearity out on one output link. In addition, the weights on the input links can be adapted based on information concerning the correctness of the output. Neural nets almost always include an inherent nonlinearity and require primarily local connectivity between nodes which are almost always nonlinear, typically analog, and may be slow compared to modern digital circuitry. Nodes may also include temporal integration and other types of time dependencies and also mathematical operations more complex than summation. [Ref. 1]

Architectures and processing elements used in neural network models are simplified versions of those observed in biological nervous systems. Figure 2 illustrates a number of different types of biological neurons and a small biological neural network. [Ref. 1]

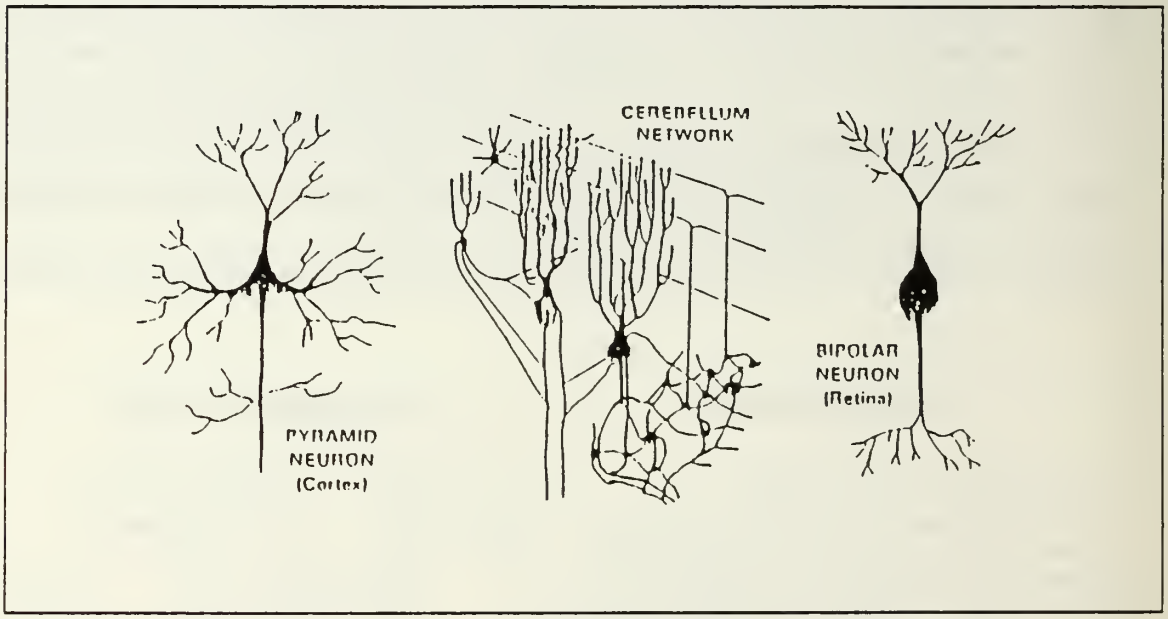


Figure 2. Biological Neurons and a Small Biological Neural Network [Ref. 1]

Characteristics of biological neural networks that artificial neural network models hope to provide include: [Ref. 1]

- Fault tolerance to loss of a small number of computational elements.
- Insensitivity to small variations between computational elements.
- The need for primarily local connectivity and local learning rules.
- Real time response.
- Parallelism.

Work in neural networks is generally oriented towards achieving rather high-level intelligent functions, such as pattern recognition, categorization, and associative memory. The biological knowledge of these functions is far from complete, but it is very clear that neurons and synapses are the fundamental devices used. It is also clear that these devices are not programmed in the conventional manner; rather, problem-specific knowledge is acquired by a learning process which alters the neuronal parameters directly. These are the two principal facts of biology that have been applied to neural networks. They are the equivalent of the transistor and of the logically structured program in conventional computers. In addition, the algorithms for calculating the output of a model neuron from its input and the high synaptic connectivity used in model networks both derive from biological observations. [Ref. 1]

Modern neuroscience provides a great wealth of additional information that has only just begun to be applied to neural network modeling. This is because the path from this more recent biological information to the desired intelligent functions is relatively tenuous, and the simple ideas of neurons, synapses and learning, are themselves surprisingly powerful. [Ref. 1]

The few principles of neurons, synapses, and learning constitute the biological foundation of most neural networks. They are, of course, insufficient to specify a network with the kinds of high-level intelligent functions mentioned above. In order to achieve these functions, the biological foundation is supplemented with cleverly invented ideas, some drawn from other disciplines, notably physics. This non-biological approach is appropriate considering the technological goals of the research, the lack of clear alternative biological solutions, and the possibility that future research will verify that such imported ideas are in fact biological. However, if biological realism is not sufficiently maintained, neural networks will lose the ability to interact profitably with neuroscience. [Ref. 1]

B. NEURAL NETWORKS IN PATTERN RECOGNITION AND CLASSIFICATION :

Pattern recognition and classification is an area where neural nets have proven to be very successful.

A taxonomy of six important neural nets that can be used for recognition and classification of unknown patterns is presented in Figure 3.

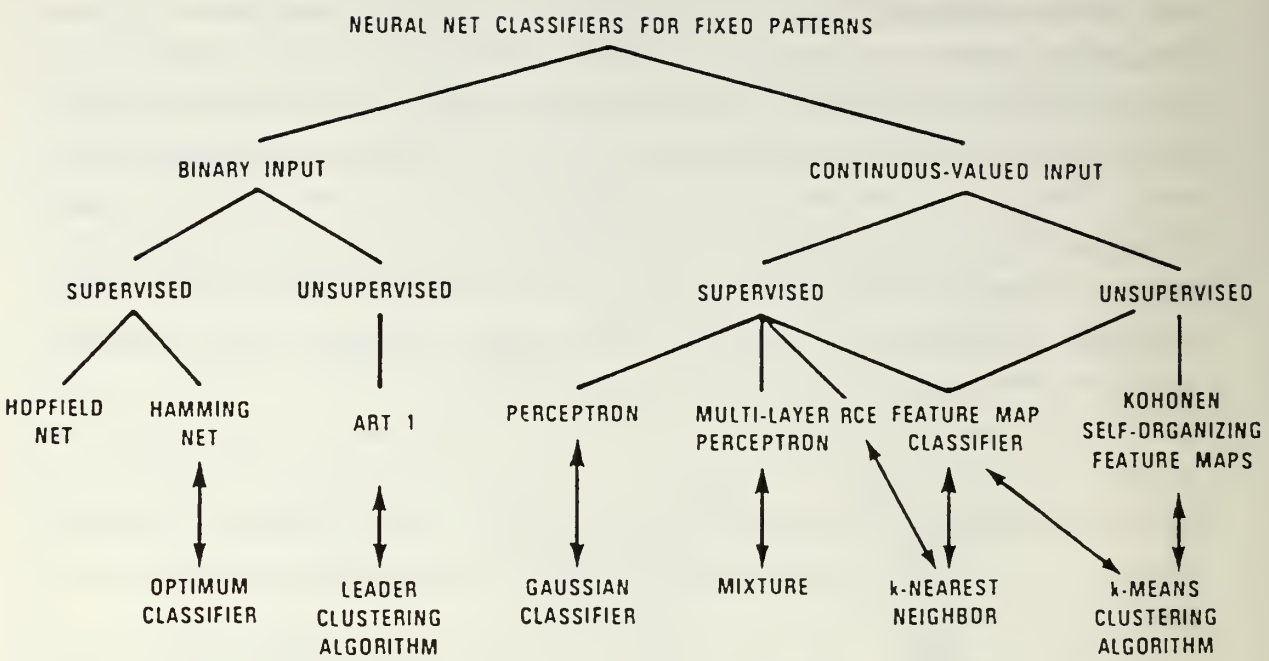


Figure 3. A taxonomy of Neural Network Classification and Clustering Models [Ref. 2]

This taxonomy is first divided between networks with binary and continuous valued inputs. Below this, nets are divided between those trained with or without supervision. Overall, adaptive neural networks can be trained using three types of training procedures: [Ref. 1]

- **Supervised training**, which requires labeled training data and an external teacher. The teacher knows the desired correct response and provides a feedback error signal after each trial. This is sometimes called *reinforcement learning*, or *learning with a critic* when the teacher only indicates whether a response was correct or incorrect and does not provide detailed error information.
- **Unsupervised training**, sometimes called *self-organization* , uses unlabeled training data and requires no external teacher. Data is presented and internal categories or clusters are formed which compress the amount of input data that must be processed at higher levels without losing important information. Clustering is an important component of many pattern classification procedures. It is sometimes called *vector quantization* when used to convert analog inputs into a binary form suitable for transmission or storage.
- **Self-supervised training** is used by automata which monitor performance internally and require no external teacher. For example, automata which learn to track a moving spot by controlling simulated eye muscles can generate an error signal

based on the distance between the position of the spot on a simulated retina and the center of fovea of the retina. Self-supervision is sometimes called *learning-by-doing* or *learning by experimentation.*

Nets trained with supervision such as the Hopfield net and perceptrons are used as associative memories or as classifiers. The teacher provides side information or labels that specify the correct class for new input patterns during training. Most traditional statistical classifiers, such as Gaussian classifiers, are trained with supervision using labeled training data [Ref. 2]. Nets trained without supervision, such as the Kohonen's feature-map forming nets [Ref. 2], are used as vector quantizers or to form clusters. The teacher does not provide these nets with any information concerning the correct class during training. The classical K-means [Ref. 2] and the leader clustering algorithm [Ref. 2] are trained without supervision.

In this study, we are going to focus on the use of neural net classifiers for fixed patterns with binary input elements. We are going to implement and simulate, for different cases of input patterns, the supervised Hopfield net, the Hamming net which is a neural net implementation of the optimum classifier for binary inputs, and the unsupervised leader clustering algorithm of the Carpenter / Grossberg net.

C. NEURAL NETWORKS AS MLSE RECEIVERS OF BINARY SIGNALS IN GAUSSIAN NOISE :

In this study, we are also going to focus on the use of neural network based maximum-likelihood sequence estimation (MLSE) receiver structure. In particular, the problem of detecting digital data symbols transmitted over a time-dispersive time-varying channel in the presence of additive Gaussian noise will be considered . We are going to computer implement this neural network structure and simulate it for stationary or time-varying transmission channel. Results of these simulations will be provided to show that neural networks offer attractive implementation alternatives for MLSE [Ref. 3].

II. THE HOPFIELD NETWORK

A. GENERALITIES:

In recent years, an upsurging interest in neural networks made of highly parallel computational elements connected in patterns that are reminiscent of biological neural nets has caught attention of researchers and scientists. In particular, more recent work has explored the ability of a neural model described by Hopfield to serve as a content-addressable memory (classifier). This network retrieves one of the M stored exemplars given an input pattern which is a noisy version of one of these exemplars. A classifier determines which of the M exemplar patterns is most similar to the noisy input pattern.

In the following study, we will focus on the classification problem because a content-addressable memory is essentially a classifier which outputs the exemplar for the selected class instead of an index to the class. Classification is a fundamental operation that is essential to the important problem of speech and image recognition whether achieved by biological or artificial means.

Past studies have demonstrated that the Hopfield model can be used as a content-addressable memory for random input patterns and to classify binary patterns created from radar cross sections, consonants and vowels extracted from spoken words, and lines in an image. These results demonstrate that a neural network based on the Hopfield model can perform classification. In addition, Hopfield models have been successfully applied to other problems, such as, the traveling salesman problem, the Analog to Digital (A-D) converter problem, and the signal decomposition problem. [Ref. 4]

B. OPERATION OF THE HOPFIELD NETWORK:

The unit used by a number of scientists is the familiar binary threshold unit (Mc Culloch-Pitts neuron) whose output is 1 if and only if $\sum_{i=1}^{N} w_{ij} s_i > \theta_j$ where $0 \leq j \leq M-1$; otherwise 0, where N is the number of elements or bits in a pattern, s_i is the current value of the ith input and w_{ij} is the corresponding synaptic weight from i to unit j whose threshold is θ_j. In the McCulloch-Pitts networks, every neuron processes its inputs to determine a new output at each time step [Ref. 5]. By contrast, a Hopfield net is a network of such units subject to the updating rule: "Pick a unit at random. If the sum of the weights on connections to other active units is positive, turn it on. Otherwise turn it off". The operation of this network is described as we first apply input values of an

unknown pattern at time zero through the bottom threshold-logic nodes. This forces the output of the net to match the unknown pattern at time zero:

$$\mu_i(0) = x_i \qquad 0 < i < N-1 \tag{2-1}$$

where $\mu_i(t)$ is the output of node i at time t and x_i is element i of the input pattern taking on the values $+1$ or -1. Following this initialization, the network iterates in discrete time steps using the given equation:

$$\mu_j(t+1) = f_h\left[\sum_{i=0}^{N-1} w_{ij}\,\mu_i(t)\right] \qquad 0 \le j \le M-1 \tag{2-2}$$

In this equation f_h is a modified hard-limiter function and w_{ij} is the weight applied to the output of node i that feeds to node j. Previousely, we have assumed the elements of the input vector x take on values $+1$ and -1, respectively, for the $+1$ and -1 states, then f_h is the symmetric hard-limiting function.

$$f_h(\alpha) = \begin{cases} +1 & , if\ \alpha > 0 \\ -1 & , if\ \alpha \le 0 \end{cases} \tag{2-3}$$

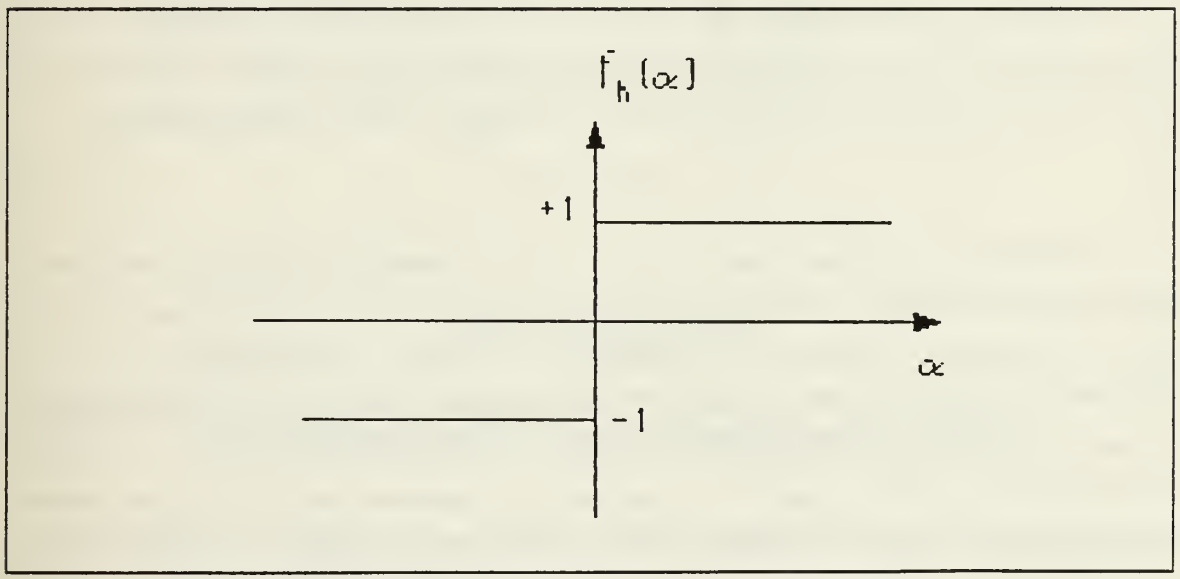


Figure 4. The hard-limiting function used in the Hopfield network

The weights are set using exemplar patterns for all stored classes.

$$w_{ij} = \begin{cases} \sum_{s=0}^{M-1} x_i^s x_j^s & i \neq j \\ 0 & i = j \quad ,0 \leq i,j \leq N-1 \end{cases} \qquad (2-4)$$

where x_i^s is element i of the exemplar for pattern s. The output of each node is fed to every other node with a weight that is symmetric, and each node does not feedback to itself. After convergence, the output of the net is the final pattern represented by the outputs of the nodes.

$$x'_i = \mu_i(\infty) \qquad i = 0,1,...,N-1 \qquad (2-5)$$

The network is considered to have converged when the outputs no longer change on successive iterations. When the Hopfield net is used as an associative memory, the network output after convergence is used directly as the restored memory. When used as a classifier, the output of the Hopfield net after convergence must be compared to the M exemplars to determine if it matches an exemplar exactly. If it does, the output is the exemplar that best matches the output pattern. If it does not, then a "no match" result occurs.

Hopfield first demonstrated that when the net is trained with M exemplar patterns using Equation 2-4, and an exemplar is presented at time zero, then the final pattern in the net after convergence will be one of the M exemplars with a high probability if,

$$M < 0.15N \qquad (2-6)$$

The exemplars thus form stable states of the net. Hopfield's statistical results were obtained with randomly generated exemplars. It is possible and relatively easy to select a set of M exemplars that satisfies Equation 2-6, but does not form stable states in the Hopfield net. These exemplars must have many elements in common. When an exemplar for one of these patterns is presented at time zero, the network does not converge to any of the trained exemplars. Instead, it converges to a spurious pattern never seen before. This problem of spurious states also occurs when a noisy exemplar is presented to the net. Even when the M exemplars are stable states of the net, there is no guarantee that noisy versions of these exemplars passed through discrete, memoryless channels and presented at time zero will converge to the original exemplars. Hopfield, for example, observed that the number of spurious states found increases substantially as more elements in the input exemplar are corrupted.

The Hopfield neural network can be used as a classifier only when:

1. The exemplars for the patterns to be classified form stable states and converge to themselves when presented at time zero as input.
2. A mechanism is provided to determine which of the M exemplars the net is closest to after convergence.

The first requirement is a necessary condition for a proper classification operation. The second is necessary because the Hopfield net by itself is not a neural-net classifier, but is more like a preprocessor which still requires a classification net to select which of the M classes an output pattern is closest to .

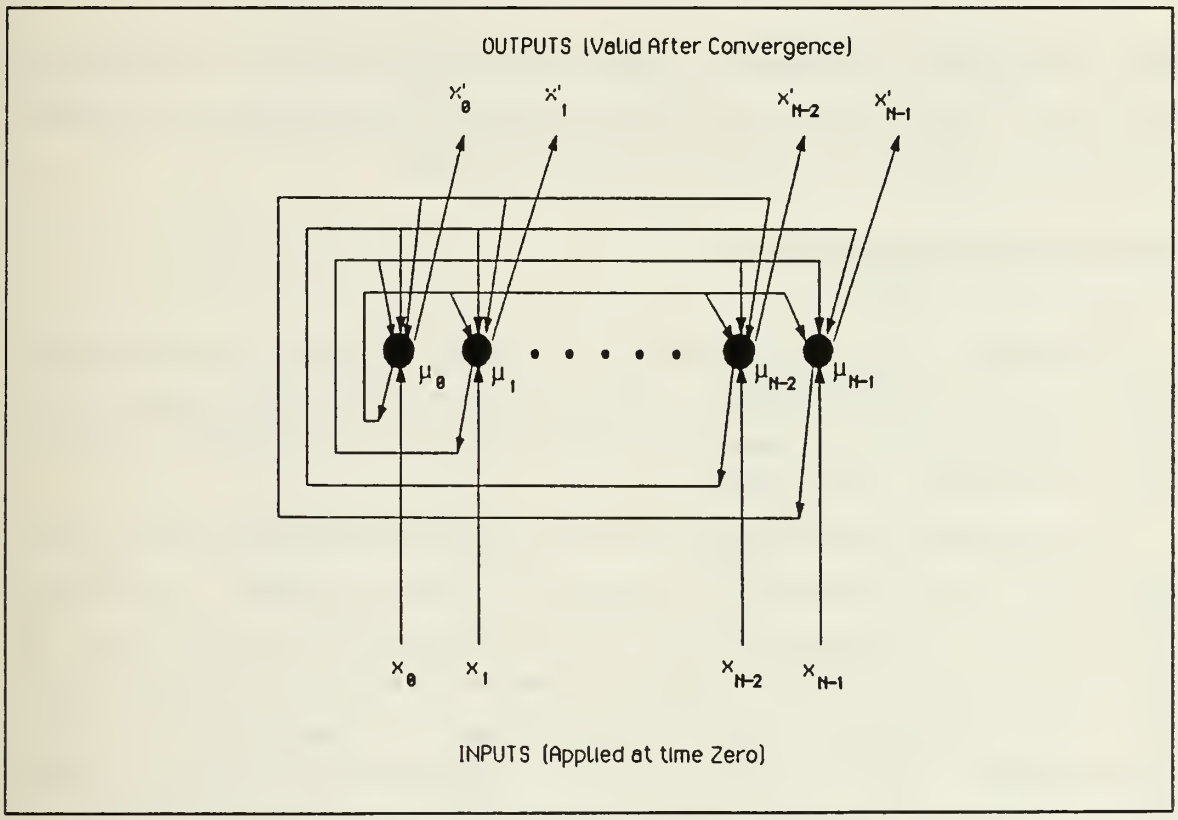


Figure 5. The Hopfield net used as a content-addressble memory [Ref. 2]

It is difficult to satisfy the requirement that exemplars form stable states without actually running the Hopfield net. In general, patterns that are more random will satisfy this requirement more easily than patterns with many bits in common.

C. IMPLEMENTATION OF THE HOPFIELD NETWORK:

The Hopfield network can be used either as an associative memory or as a content-addressable memory which is described in this study. The Hopfield net shown in Figure

5, has N nodes containing hard-limiting nonlinearities and binary inputs and outputs taking on the values +1 and −1. The output of each node is fed back to all other nodes via weights denoted w_{ij}.

A computer algorithm to implement the operation of this net as a content-addressable memory can be summarized in four necessary steps : [Ref. 2]

Step 1. Assign Connection weights : using Equation 2-4

Step 2. Initialize with Unknown Input Pattern : using Equation 2-1

Step 3. Iterate Until Convergence : using Equation 2-2, the process is repeated until node outputs remain unchanged with further iterations. The node outputs then represent the exemplar pattern that best matches the unknown input.

Step 4. Repeat by Going to Step 2

The weights are first set using Equation 2-4 and elements of the M stored exemplar patterns as the operation algorithm of the net stated in the first step. Eight patterns (M = 8), shown in Figure 6, have been selected to simulate this algorithm and were stored in the memory of the network.

For convenience, these eight patterns were selected to be 120 nodes (12 by 10 matrices) each. The only limitation in the choice of N (number of nodes) is the time that the net will take to iterate and converge to an output pattern or respond with a "no match". After assigning connection weights, an unknown input pattern is imposed on the net at time zero by forcing the output of the net to match the unknown pattern. Then, the net iterates in discrete time steps. The net is considered to have converged when outputs no longer change with further iterations. The pattern specified by the node outputs, after convergence is reached, is the net output. [Ref. 4]

Using the Hopfield net as a classifier, the output will be compared to every one of the M class patterns. If the output matches an exemplar, the classification is terminated and the output is that class whose exemplar best matches the output pattern. If it does not, then a "no match" result occurs.

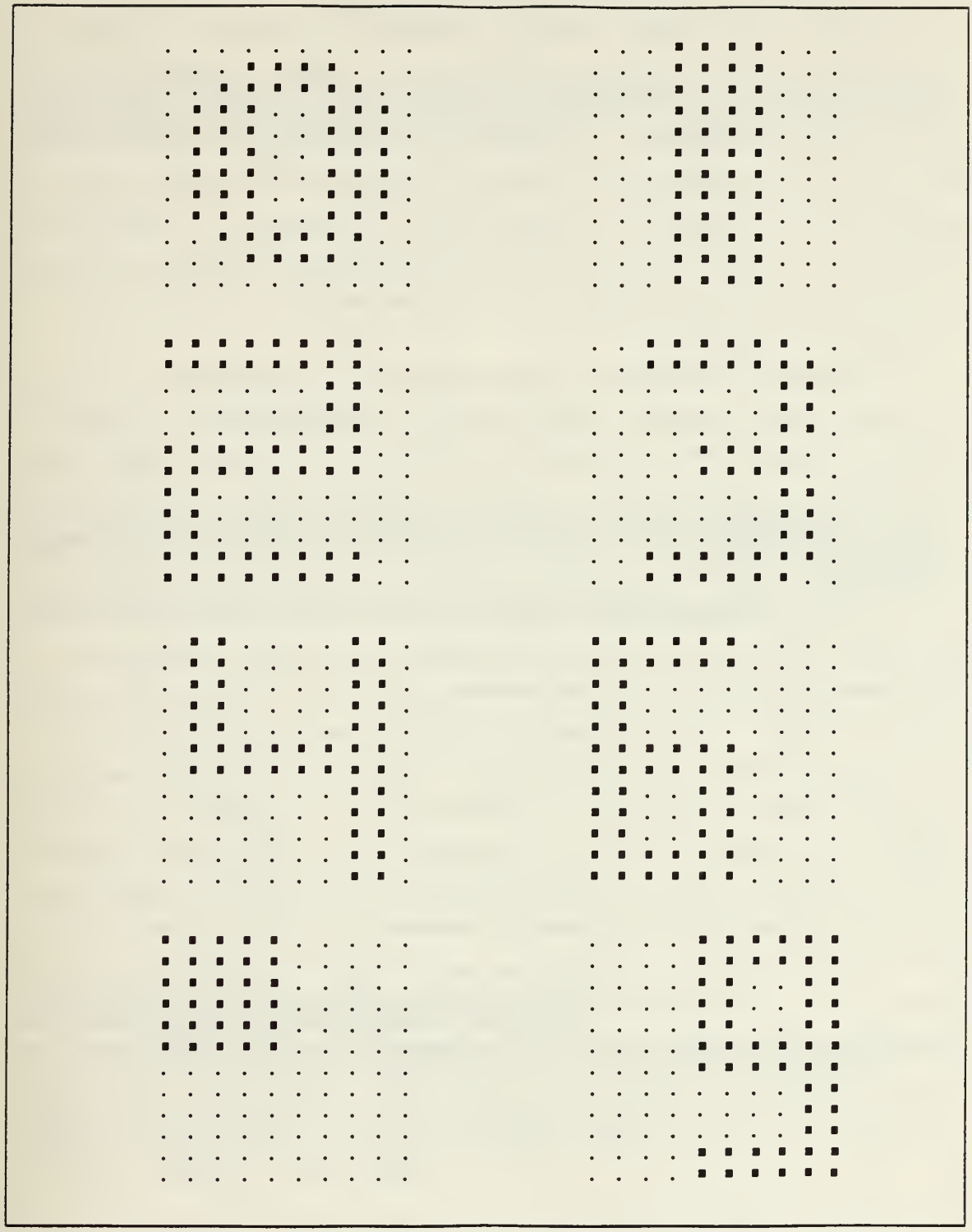


Figure 6. The Eight stored patterns

D. SIMULATION OF THE HOPFIELD NETWORK:

Using the Fortran program provided in Appendix A, we simulated the operation of the Hopfield net when used as an addressable-content memory (classifier) for different inputs. For convenience and simplicity, the M class patterns were first introduced to the net as a matrix of N by M (120 by 8), where each vector colunm of 120 nodes represents the 12 by 10 matrix representation of the class. The first vector colunm represents the pattern of a '0', the second of a '1', the third of a '2', the fourth of a '3', the fifth of a '4', the sixth of a '6', the seventh of the block pattern representation of the 'point' and the eight and last of a '9'. The elements of each class pattern take on the values of +1 for a ' black pixel ' and −1 for a ' . '.

The behavior of the network is simulated first by presenting the pattern of digit '3' as an input pattern. To make it more interesting, a corrupted version of this pattern is achieved by randomly reversing each bit, of the matrix representation of digit 3, independently from +1 to −1 and vice versa with a probability of 0.25. Implementing this pattern is equivalent to receiving noise corrupted bits of a digit in a noisy communication channel.

The corrupted input pattern was then imposed on the net at time zero. After the first iteration, the net still can not tell which class the input pattern corresponds to. As more iterations took place, the output becomes more and more like the correct exemplar pattern of the digit 3, as you can see in this simulation result provided in Figure 7.

Then, at the third iteration, the net has converged and the output, as can be seen, is indeed the pattern of the digit 3. Only three iterations were sufficient for the net to converge to the corrected digit and to recognize it as a 3. Now, we tried to see if we present a corrupted pattern of another digit, how many iterations will be used to converge to the correct result? Will it take only three iterations to do so ? The input pattern that we used was of digit '9' and using the same procedure we randomly reversed each bit from +1 to −1 and vice versa with the same probability and error distribution as before. The response of the network to this input pattern is illustrated in Figures 8 and 9.

THE UNKNOWN INPUT PATTERN TO THE HOPFIELD NETWORK, ON THE LEFT AS IMPOSED ON THE NETWORK AND ON THE RIGHT IN A MUCH CLEAR REPRESENTATION WHERE EVERY (▪) REPLACES A 1 AND EVERY (.) REPLACES A -1:

```
-1 -1 -1  1  1  1  1 -1 -1  1      . . . ▪ ▪ ▪ ▪ . . ▪
-1  1 -1  1  1  1  1  1  1  1      . ▪ . ▪ ▪ ▪ ▪ ▪ ▪ ▪
 1 -1 -1 -1  1 -1 -1  1 -1 -1      ▪ . . . ▪ . . ▪ . .
 1 -1 -1 -1 -1 -1 -1 -1  1 -1      ▪ . . . . . . . ▪ .
-1 -1 -1 -1 -1  1 -1 -1  1 -1      . . . . . ▪ . . ▪ .
-1 -1  1 -1  1  1  1  1 -1 -1      . . ▪ . ▪ ▪ ▪ ▪ . .
 1 -1 -1 -1  1  1 -1 -1 -1 -1      ▪ . . . ▪ ▪ . . . .
-1 -1 -1  1 -1 -1 -1 -1  1  1      . . . ▪ . . . . ▪ ▪
-1 -1 -1 -1 -1 -1  1  1  1 -1      . . . . . . ▪ ▪ ▪ .
-1 -1 -1  1 -1 -1 -1  1 -1 -1      . . . ▪ . . . ▪ . .
 1  1  1  1  1 -1  1 -1 -1 -1      ▪ ▪ ▪ ▪ ▪ . ▪ . . .
-1 -1 -1  1  1  1 -1  1 -1  1      . . . ▪ ▪ ▪ . ▪ . ▪
```

THE OUTPUT OF THE HOPFIELD NETWORK LOOKS LIKE THE FOLLOWING FOR THE UNKNOWN INPUT PATTERN PRESENTED. THE PATTERN ON THE LEFT CORRESPONDS TO THE OUTPUT AFTER THE 1ST ITERATION WHILE THE PATTERN IN THE MIDDLE CORRESPONDS TO THE OUTPUT AFTER THE 2ND ITERATION, THE PATTERN ON THE RIGHT CORRESPONDS TO THE OUTPUT AFTER THE 3RD ITERATION.

```
. . . ▪ ▪ ▪ ▪ . . ▪      . . ▪ ▪ ▪ ▪ ▪ ▪ . .      . . ▪ ▪ ▪ ▪ ▪ ▪ . .
. ▪ . ▪ ▪ ▪ ▪ ▪ ▪ ▪      . . ▪ ▪ ▪ ▪ ▪ ▪ ▪ .      . . ▪ ▪ ▪ ▪ ▪ ▪ ▪ .
▪ . . . ▪ . . ▪ . .      . . . . . . . ▪ ▪ .      . . . . . . . ▪ ▪ .
▪ . . . . . . . ▪ .      . . . . . . . ▪ ▪ .      . . . . . . . ▪ ▪ .
. . . . . ▪ . . ▪ .      . . . . . . . ▪ ▪ .      . . . . . . . ▪ ▪ .
. . ▪ . ▪ ▪ ▪ ▪ . .      . . . . ▪ ▪ ▪ ▪ . .      . . . . ▪ ▪ ▪ ▪ . .
▪ . . . ▪ ▪ . . . .      . . . . ▪ ▪ ▪ ▪ . .      . . . . ▪ ▪ ▪ ▪ . .
. . . ▪ . . . . ▪ ▪      . . . . . . . . ▪ .      . . . . . . . ▪ ▪ .
. . . . . . ▪ ▪ ▪ .      . . . . . . . . ▪ .      . . . . . . . ▪ ▪ .
. . . ▪ . . . ▪ . .      . . . . . . . . ▪ .      . . . . . . . ▪ ▪ .
▪ ▪ ▪ ▪ ▪ . ▪ . . .      . . ▪ ▪ ▪ ▪ ▪ ▪ ▪ .      . . ▪ ▪ ▪ ▪ ▪ ▪ ▪ .
. . . ▪ ▪ ▪ . ▪ . ▪      . . ▪ ▪ ▪ ▪ ▪ ▪ . .      . . ▪ ▪ ▪ ▪ ▪ ▪ . .
```

CLASSIFICATION OF THE UNKNOWN INPUT PATTERN:

AT THIS POINT, FURTHER ITERATIONS WON'T MAKE ANY CHANGE ON THE OUTPUT OF THE NETWORK AND THE PATTERN SPECIFIED BY THE OUTPUT NODES IS THE NET'S OUTPUT. THE TASK OF THE NET NOW IS TO CLASSIFY THE INPUT AS AN ALREADY KNOWN PATTERN OR A NO MATCH WILL OCCUR.
AFTER CLASSIFICATION, THE OUTPUT PATTERN OF THE HOPFIELD NET MATCHES BEST THE PATTERN OF DIGIT THREE.

Figure 7. Hopfielf net response to the first input pattern

THE UNKNOWN INPUT PATTERN TO THE HOPFIELD NETWORK, ON THE LEFT AS IMPOSED ON THE NETWORK AND ON THE RIGHT IN A MUCH CLEAR REPRESENTATION WHERE EVERY (▪) REPLACES A 1 AND EVERY (.) REPLACES A -1:

```
-1 -1  1 -1  1  1  1 -1  1 -1      . .  ▪  .  ▪  ▪  ▪  .  ▪  .
-1  1  1 -1  1  1  1  1  1 -1      .  ▪  ▪  .  ▪  ▪  ▪  ▪  ▪  .
 1 -1 -1 -1 -1  1 -1 -1 -1  1      ▪  .  .  .  .  ▪  .  .  .  ▪
 1 -1 -1 -1  1  1 -1  1  1  1      ▪  .  .  .  ▪  ▪  .  ▪  ▪  ▪
-1 -1 -1 -1  1 -1 -1  1  1  1      .  .  .  .  ▪  .  .  ▪  ▪  ▪
-1 -1  1 -1  1  1  1  1  1  1      .  .  ▪  .  ▪  ▪  ▪  ▪  ▪  ▪
 1 -1 -1 -1  1  1 -1 -1  1  1      ▪  .  .  .  ▪  ▪  .  .  ▪  ▪
-1 -1 -1  1 -1 -1 -1  1  1 -1      .  .  .  ▪  .  .  .  ▪  ▪  .
-1 -1 -1 -1 -1 -1  1 -1  1  1      .  .  .  .  .  .  ▪  .  ▪  ▪
-1 -1 -1  1 -1 -1 -1 -1 -1  1      .  .  .  ▪  .  .  .  .  .  ▪
 1  1 -1 -1  1 -1  1 -1 -1  1      ▪  ▪  .  .  ▪  .  ▪  .  .  ▪
-1 -1  1 -1  1  1 -1  1  1 -1      .  .  ▪  .  ▪  ▪  .  ▪  ▪  .
```

THE OUTPUT OF THE HOPFIELD NETWORK LOOKS LIKE THE FOLLOWING FOR THE UNKNOWN INPUT PATTERN PRESENTED. THE PATTERN ON THE LEFT CORRESPONDS TO THE OUTPUT AFTER THE 1ST ITERATION WHILE THE PATTERN ON THE RIGHT CORRESPONDS TO THE OUTPUT AFTER THE 2ND ITERATION.

```
. .  ▪  .  ▪  ▪  ▪  .  ▪  .          .  .  .  .  ▪  ▪  ▪  ▪  ▪  ▪
.  ▪  ▪  .  ▪  ▪  ▪  ▪  ▪  .          .  .  .  .  ▪  ▪  ▪  ▪  ▪  ▪
▪  .  .  .  .  ▪  .  .  .  ▪          .  .  .  .  ▪  ▪  ▪  .  ▪  ▪
▪  .  .  .  ▪  ▪  .  ▪  ▪  ▪          .  .  .  .  ▪  ▪  ▪  .  .  ▪
.  .  .  .  ▪  .  .  ▪  ▪  ▪          .  .  .  .  ▪  ▪  ▪  .  .  ▪
.  .  ▪  .  ▪  ▪  ▪  ▪  ▪  ▪          .  .  .  .  ▪  ▪  ▪  ▪  ▪  ▪
▪  .  .  .  ▪  ▪  .  .  ▪  ▪          .  .  .  .  ▪  ▪  ▪  ▪  ▪  ▪
.  .  .  ▪  .  .  .  ▪  ▪  .          .  .  .  .  .  .  .  .  ▪  ▪
.  .  .  .  .  .  .  ▪  .  ▪          .  .  .  .  .  .  .  .  ▪  ▪
.  .  .  ▪  .  .  .  .  .  ▪          .  .  .  .  .  .  .  .  ▪  ▪
▪  ▪  .  .  ▪  .  ▪  .  .  ▪          .  .  .  .  ▪  ▪  ▪  ▪  ▪  ▪
.  .  ▪  .  ▪  ▪  .  ▪  ▪  .          .  .  .  .  ▪  ▪  ▪  ▪  ▪  ▪
```

THE OUTPUT OF THE HOPFIELD NETWORK AFTER THE 3RD ITERATION IS PRESENTED ON THE LEFT WHILE THE PATTERN ON THE RIGHT CORRESPONDS TO THE OUTPUT AFTER THE 4TH ITERATION.

```
.  .  .  .  ▪  ▪  ▪  ▪  ▪  ▪          .  .  .  .  ▪  ▪  ▪  ▪  ▪  ▪
.  .  .  .  ▪  ▪  ▪  ▪  ▪  ▪          .  .  .  .  ▪  ▪  ▪  ▪  ▪  ▪
.  .  .  .  ▪  ▪  .  .  ▪  ▪          .  .  .  .  ▪  ▪  .  ▪  ▪  ▪
.  .  .  .  ▪  ▪  .  .  ▪  ▪          .  .  .  .  ▪  ▪  .  ▪  ▪  ▪
.  .  .  .  ▪  ▪  .  .  ▪  ▪          .  .  .  .  ▪  ▪  .  ▪  ▪  ▪
.  .  .  .  ▪  ▪  ▪  ▪  ▪  ▪          .  .  .  .  ▪  ▪  ▪  ▪  ▪  ▪
.  .  .  .  ▪  ▪  ▪  ▪  ▪  ▪          .  .  .  .  ▪  ▪  ▪  ▪  ▪  ▪
.  .  .  .  .  .  .  .  ▪  ▪          .  .  .  .  .  .  .  .  ▪  ▪
.  .  .  .  .  .  .  .  ▪  ▪          .  .  .  .  .  .  .  .  ▪  ▪
.  .  .  .  .  .  .  .  ▪  ▪          .  .  .  .  .  .  .  .  ▪  ▪
.  .  .  .  ▪  ▪  ▪  ▪  ▪  ▪          .  .  .  .  ▪  ▪  ▪  ▪  ▪  ▪
.  .  .  .  ▪  ▪  ▪  ▪  ▪  ▪          .  .  .  .  ▪  ▪  ▪  ▪  ▪  ▪
```

Figure 8. Hopfield net response to the second input pattern

```
CLASSIFICATION OF THE UNKNOWN INPUT PATTERN:
============================================
AT THIS POINT, FURTHER ITERATIONS WON'T MAKE ANY CHANGE ON THE OUTPUT
OF THE NETWORK AND THE PATTERN SPECIFIED BY THE OUTPUT NODES IS THE
NET'S OUTPUT. THE TASK OF THE NET NOW IS TO CLASSIFY THE INPUT AS AN
ALREADY KNOWN PATTERN OR A NO MATCH WILL OCCUR.
AFTER CLASSIFICATION, THE OUTPUT PATTERN OF THE HOPFIELD NET MATCHES
BEST THE PATTERN OF DIGIT NINE.
```

Figure 9. Classification response of the Hopfield net to the second input pattern

Then, using the same probability of error and distribution of errors in the input pattern as for digit '3', we conclude from the result of this simulation that the number of iterations needed to get the right answer depends on how close the input pattern was to a stored one. This may explain why there is a difference in the number of iterations taken by the net to converge to the right answer. We may say that the corrupted input pattern of digit '3' was closer to the perfect exemplar for the '3' than the noise disturbed input pattern of digit '9' is to the perfect '9' using the same noise corrupted bits distribution and the same probability of error as for digit '3'.

Using the noise disturbed pattern for digit '9' as input to the net, but now with different error distribution and the same probability of error (0.25), the simulation and results were as shown in Figure 10.

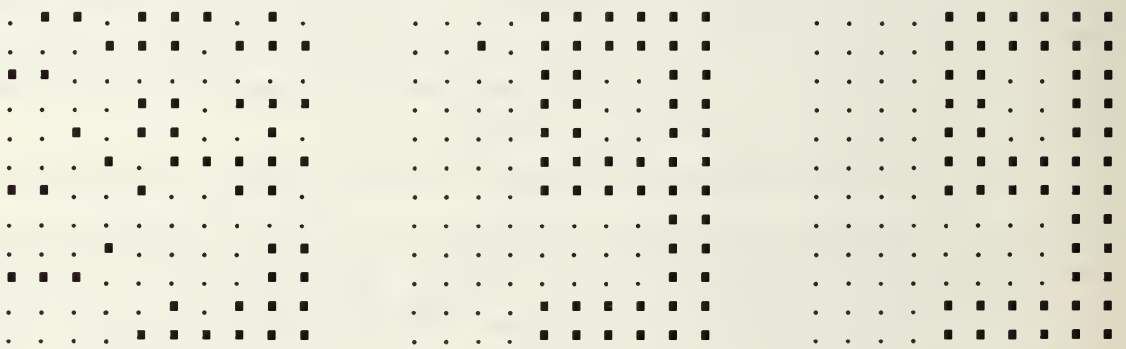
The result shows perfectly that the number of iterations that the network needs to converge depends on the error distribution in the input pattern and not on the probability of error.

Now as a conclusion after all these simulations and results, one could say that when imposing a perfect input pattern on the net, the network will take only one iteration to recognize it as a stored one. To verify this finding, we take the pattern of digit '2' and present it as the input pattern to the net at time zero without making any change in its elements (a perfect exemplar of digit '2'). The response of the net to this perfect input pattern is presented in Figure 11.

THE UNKNOWN INPUT PATTERN TO THE HOPFIELD NETWORK, ON THE LEFT AS IMPOSED ON THE NETWORK AND ON THE RIGHT IN A MUCH CLEAR REPRESENTATION WHERE EVERY (▪) REPLACES A 1 AND EVERY (.) REPLACES A -1:

```
-1  1  1 -1  1  1  1 -1  1 -1        .  ▪  ▪  .  ▪  ▪  ▪  .  ▪  .
-1 -1 -1  1  1  1 -1  1  1  1        .  .  .  ▪  ▪  ▪  .  ▪  ▪  ▪
 1  1 -1 -1 -1 -1 -1 -1 -1 -1        ▪  ▪  .  .  .  .  .  .  .  .
-1 -1 -1 -1  1  1 -1  1  1  1        .  .  .  .  ▪  ▪  .  ▪  ▪  ▪
-1 -1  1 -1  1  1 -1 -1  1 -1        .  .  ▪  .  ▪  ▪  .  .  ▪  .
-1 -1 -1  1 -1  1  1  1  1  1        .  .  .  ▪  .  ▪  ▪  ▪  ▪  ▪
 1  1 -1 -1  1 -1 -1  1  1 -1        ▪  ▪  .  .  ▪  .  .  ▪  ▪  .
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1        .  .  .  .  .  .  .  .  .  .
-1 -1 -1  1 -1 -1 -1 -1  1  1        .  .  .  ▪  .  .  .  .  ▪  ▪
 1  1  1 -1 -1 -1 -1 -1  1  1        ▪  ▪  ▪  .  .  .  .  .  ▪  ▪
-1 -1 -1 -1 -1  1 -1  1  1  1        .  .  .  .  .  ▪  .  ▪  ▪  ▪
-1 -1 -1 -1  1  1  1  1  1  1        .  .  .  .  ▪  ▪  ▪  ▪  ▪  ▪
```

THE OUPUT OF THE HOPFIELD NETWORK LOOKS LIKE THE FOLLOWING FOR THE UNKNOWN INPUT PATTERN PRESENTED. THE PATTERN ON THE LEFT CORRESPONDS TO THE OUTPUT AFTER THE 1ST ITERATION WHILE THE PATTERN IN THE MIDDLE CORRESPONDS TO THE OUTPUT AFTER THE 2ND ITERATION, THE PATTERN ON THE RIGHT CORRESPONDS TO THE OUTPUT AFTER THE 3RD ITERATION.



CLASSIFICATION OF THE UNKNOWN INPUT PATTERN:

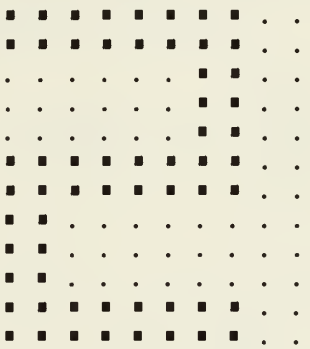
AT THIS POINT, FURTHER ITERATIONS WON'T MAKE ANY CHANGE ON THE OUTPUT OF THE NETWORK AND THE PATTERN SPECIFIED BY THE OUTPUT NODES IS THE NET'S OUTPUT. THE TASK OF THE NET NOW IS TO CLASSIFY THE INPUT AS AN ALREADY KNOWN PATTERN OR A NO MATCH WILL OCCUR.
AFTER CLASSIFICATION, THE OUTPUT PATTERN OF THE HOPFIELD NET MATCHES BEST THE PATTERN OF DIGIT NINE.

Figure 10. Hopfield net response to the third input pattern

THE UNKNOWN INPUT PATTERN TO THE HOPFIELD NETWORK, ON THE LEFT AS IMPOSED ON THE NETWORK AND ON THE RIGHT IN A MUCH CLEAR REPRESENTATION WHERE EVERY (▪) REPLACES A 1 AND EVERY (.) REPLACES A -1:

```
 1  1  1  1  1  1  1  1 -1 -1      ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ . .
 1  1  1  1  1  1  1  1 -1 -1      ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ . .
-1 -1 -1 -1 -1 -1  1  1 -1 -1      . . . . . . ▪ ▪ . .
-1 -1 -1 -1 -1 -1  1  1 -1 -1      . . . . . . ▪ ▪ . .
-1 -1 -1 -1 -1 -1  1  1 -1 -1      . . . . . . ▪ ▪ . .
 1  1  1  1  1  1  1  1 -1 -1      ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ . .
 1  1  1  1  1  1  1  1 -1 -1      ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ . .
 1  1 -1 -1 -1 -1 -1 -1 -1 -1      ▪ ▪ . . . . . . . .
 1  1 -1 -1 -1 -1 -1 -1 -1 -1      ▪ ▪ . . . . . . . .
 1  1 -1 -1 -1 -1 -1 -1 -1 -1      ▪ ▪ . . . . . . . .
 1  1  1  1  1  1  1  1 -1 -1      ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ . .
 1  1  1  1  1  1  1  1 -1 -1      ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ . .
```

THE OUTPUT OF THE HOPFIELD NETWORK LOOKS LIKE THE FOLLOWING FOR THE UNKNOWN INPUT PATTERN PRESENTED. THE PATTERN SHOWN HERE IS THE NET'S OUTPUT AFTER THE 1ST ITERATION.



CLASSIFICATION OF THE UNKNOWN INPUT PATTERN:

AT THIS POINT, FURTHER ITERATIONS WON'T MAKE ANY CHANGE ON THE OUTPUT OF THE NETWORK AND THE PATTERN SPECIFIED BY THE OUTPUT NODES IS THE NET'S OUTPUT. THE TASK OF THE NET NOW IS TO CLASSIFY THE INPUT AS AN ALREADY KNOWN PATTERN OR A NO MATCH WILL OCCUR.
AFTER CLASSIFICATION, THE OUTPUT PATTERN OF THE HOPFIELD NET MATCHES BEST THE PATTERN OF DIGIT TWO.

Figure 11. Hopfield net response to the perfect input pattern

The result was as we thought; it took the net only one iteration to recognize the input pattern as one of the M class stored patterns. From these simulations we can conclude that the Hopfield network when used as a classifier can be useful in a communication receiver where its task is to recognize the received bits, also to correct the corrupted ones and to recognize them. However, the Hopfield net is only iterating between an input pattern and the ones that are already stored in the memory of the net. The number of these patterns (M) is a limitation to the proper operation of the net as a classifier because of the convergence condition demonstrated by Hopfield, which states that the net will converge with high probability if $M < 0.15N$. A non-learning network is what the Hopfield net is, compared to other networks that we are going to describe in the following chapters. However, it has an important advantage over the others, its ability to recognize patterns even in noisy environement as long as the original pattern was stored in its memory prior to its use, otherwise, a "no match" will occur.

III. THE HAMMING NETWORK

A. GENERALITIES :

The Hopfield net, as we have seen, is often tested on problems of pattern recognition and classification by taking an input exemplar and reversing its bits randomly with a certain probability. The classifier in this study will calculate the Hamming distance to the exemplar of each class and select that class with the minimum Hamming distance to the specified input pattern. The Hamming distance is the number of bits in the input which do not match the corresponding exemplar bits. A net, which will be called the Hamming net, implements this algorithm using neural net component. Instead of calculating the Hamming distance directly, we will calculate N minus the Hamming distance and maximize this function, where N is the number of elements or bits in a pattern representation. [Ref. 4]

N minus the Hamming distance can be calculated from a weighted sum of the N elements of the input vector. If the elements of the input pattern to the net take on the values +1 and −1 for the respective states, then

$$N - N_{ham}^{j} = c_j + \sum_{i=0}^{N-1} w_{ij} x_i \qquad (3-1)$$

where,

$$w_{ij} = \frac{x_i^j}{2} \qquad (3-2)$$

and

$$c_j = \frac{N}{2} \qquad (3-3)$$

Here x_i^j is the value of element i of the exemplar for class j. When all elements in the input vector match an exemplar exactly, each element in the sum of Equation 3-1 adds $\frac{1}{2}$, and adding c_j given in Equation 3-3 gives a total of N. Whenever an element in the input pattern does not match the corresponding element in the exemplar, the prior total is decremented by 1 as required. [Ref. 4]

On the other hand, when elements of the input pattern x take on the values 0 and +1 for the −1 and +1 states, respectively, N minus the Hamming distance can be calculated from :

$$N - N_{ham}^{j} = c_j + \sum_{i=0}^{N-1} w_{ij} x_i \qquad (3-4)$$

where,

$$w_{ij} = \begin{cases} +1 & if\ x_i^j = +1 \\ -1 & if\ x_i^j = 0 \end{cases} \qquad (3-5)$$

and

$$c_j = N_z^j = N - \sum_{i=0}^{N-1} x_i^j \qquad (3-6)$$

In the above equation, N_z^j represents the number of zero elements in the exemplar for class j. When all elements in the input pattern match an exemplar exactly, the sum in Equation 3-4 adds up to the number of positive input elements. This is added to the number of zero input elements results in N, as desired. The sum is reduced by one whenever a zero input element that matches an exemplar becomes positive, and whenever a positive input element that matches an exemplar becomes zero. [Ref. 4]

Here we have made a brief introduction to the Hamming net used as a classifier, as the net that calculates the Hamming distance to exemplars for all classes and then select that class which produces the minimum Hamming distance to the input pattern. Also , we have introduced two kinds of input patterns, one which elements take on +1 and −1 values, the other +1 and 0 values, for the +1 and −1 states, respectively. Its consequences on the calculation of N minus the Hamming distance were also introduced. The next paragraph will discuss the Hamming net in further detail and illustrate how the selection of the minimum is made.

B. OPERATION OF THE HAMMING NETWORK :

Two neural nets that are logically required to implement an optimum classifier for binary patterns will be assembled to form the Hamming net. One net forms the weighted

sum to calculate quantities related to the likelihood of the different classes. The second net picks the maximum. [Ref. 4]

The first net that forms weighted sums is presented in Figure 12. An input pattern **x** is applied at the bottom of this net and an output pattern **y** is produced at the top.

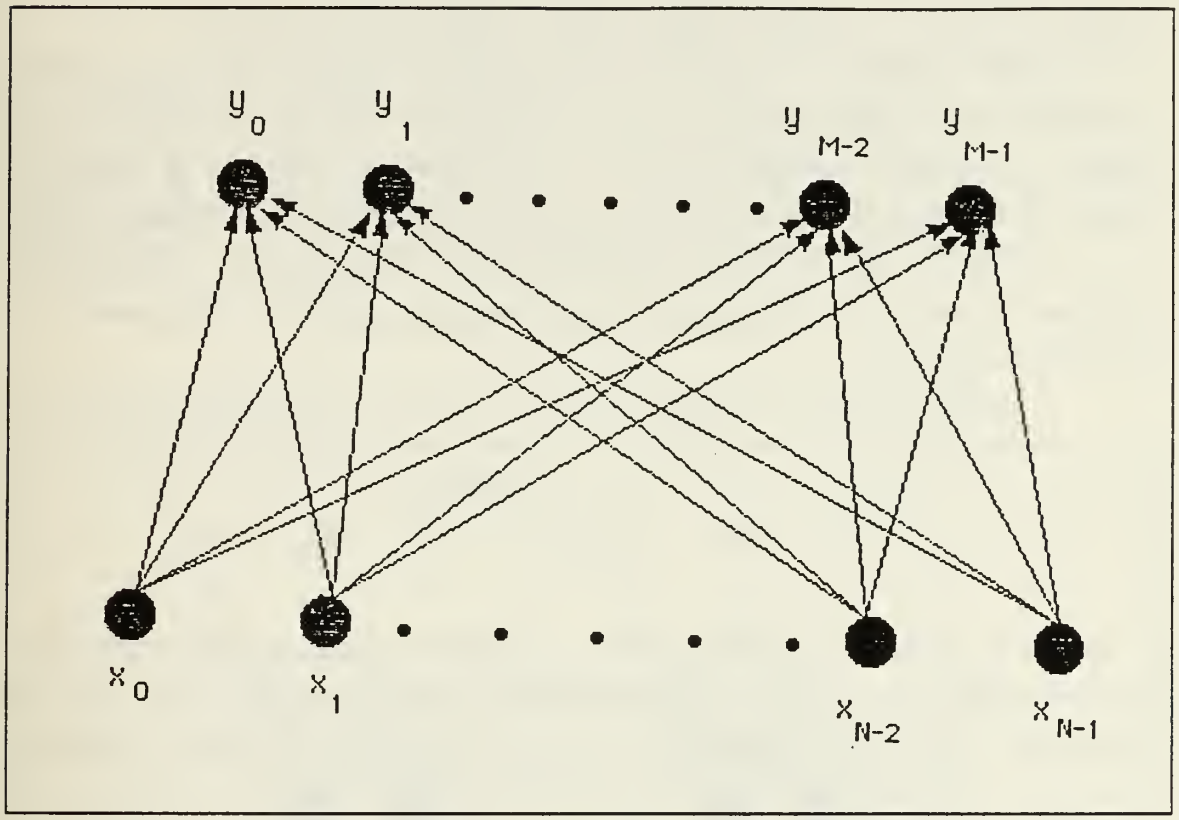


Figure 12. Feed-forward neural net used to calculate M weighted sums from the N elements of the input pattern [Ref. 4]

The first layer of nodes sends values of the input pattern to the links feeding the second layer. The second layer of nodes uses nonlinear threshold logic elements to sum weighted values of the inputs and add internal offsets [Ref. 4]. Output values from the second layer are

$$y_j = f_t\left(c_j + \sum_{i=0}^{N-1} w_{ij}\, x_i\right) \qquad 0 \le j \le M-1 \qquad (3-7)$$

where,

$$f_t(\alpha) = \begin{cases} \alpha & if\ \alpha > 0 \\ 0 & if\ \alpha \leq 0 \end{cases} \qquad (3-8)$$

In these equations, $f_t(\alpha)$ is a nonlinear function that models the nonlinearity inherent in a biological neuron. c_j is an internal offset associated with each threshold logic node, and w_{ij} are positive or negative weights associated with the links [Ref. 4].

A number of different nets can be used to pick the maximum value from the y_j outputs of the feed-forward neural net shown in Figure 12. In situations where it is only important to know if the input matches a stored state very closely, it is sufficient to identify those second-level nodes in Figure 12, with output values that exceed a specified threshold. This can be performed by modifying the constant c_j added in Equation 3-7 such that only the output of those nodes corresponding to closely matching stored states are positive. [Ref. 4]

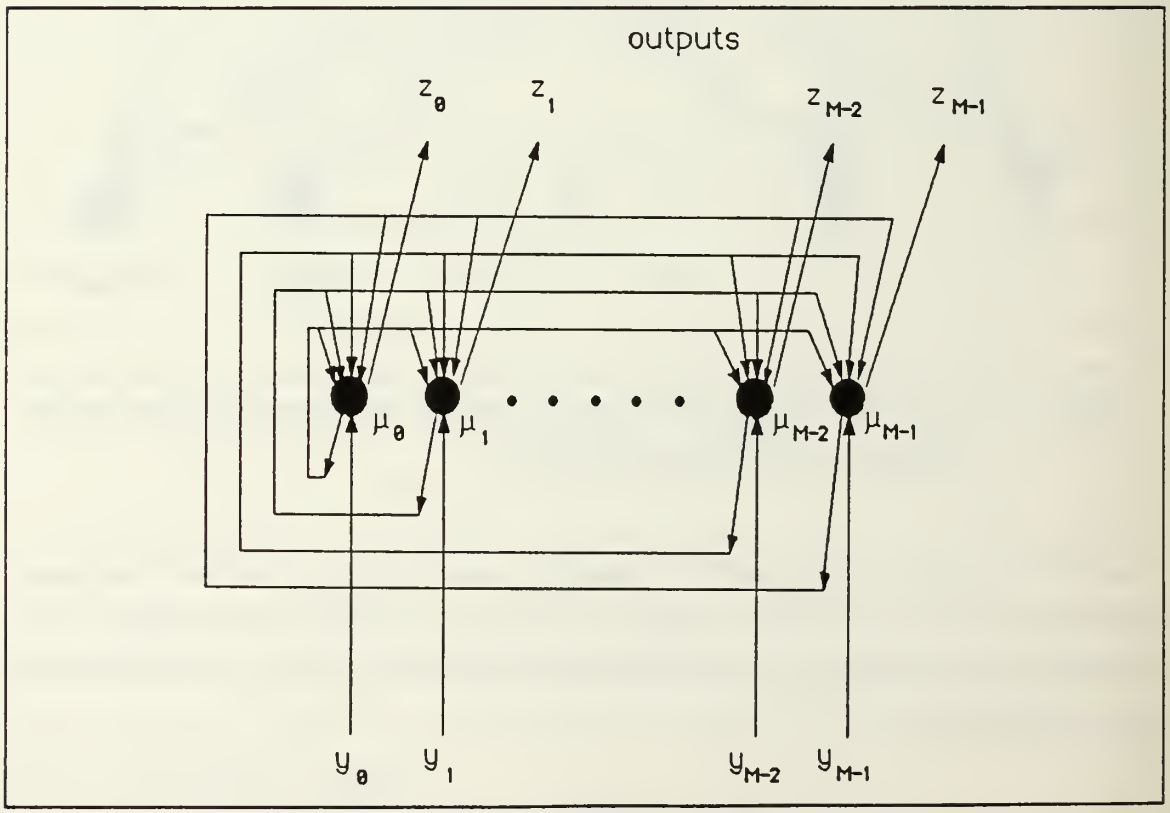


Figure 13. The iterative neural net called "maxnet" that picks the maximum of M inputs [Ref. 4]

In the more general situation, a net must select the maximum over the My_j values. There are many different neural net structures which perform this task. A less complex net that uses feedback connections to pick the maximum output y_j (referred to as a **maxnet**) is presented in Figure 13. [Ref. 4]

Although this net is similar in structure to the Hopfield net (Figure 5), it uses threshold-logic nodes, relative to the threshold-logic nonlinear function described in Equation 3-8, rather than hard-limiting nodes and feeds the output of each node back to its input instead of disallowing this feedback path. The maxnet is a fully connected net made up of only M threshold logic nodes with internal thresholds set to zero. Input values are applied at time zero through the input nodes on the bottom of Figure 13. This initializes node outputs for each node at time zero [$\mu_j(0)$] to the input values :

$$\mu_j(0) = y_j \quad j = 0,1,...,M-2,M-1 \tag{3-9}$$

The network then iterates to find the maximum using the following equation :

$$\mu_j(t+1) = f_t\left[\mu_j(t) - \varepsilon \sum_{i \neq j} \mu_i(t)\right] \quad 0 \leq i, j \leq M-1 \tag{3-10}$$

In this equation, f_t is the threshold logic function described in Equation 3-8. Each node inhibits all other nodes with a value equal to the node's output multiplied by a small negative weight (ε) which is less than $\frac{1}{M}$. Each node also feeds back to itself with unity gain. After convergence, only that output node corresponding to the maximum input will have a nonzero value. This value will generally be less than the original time zero value of that node. The output values of the net are thus simply the node output values after convergence :

$$z_j = \mu_j(\infty) \quad j = 0,1,...,M-2,M-1 \tag{3-11}$$

The maxnet will converge and find the maximum input when

$$w_{ij} < \frac{1}{M-1} \tag{3-12}$$

By convergence, we mean that the output nodes stop changing in time and only the output of one node corresponding to the maximum input is positive. Applying the threshold logic function f_t on each one of these output nodes will result in only one

nonzero output node, the previous positive one, and a zero value for all the others. The nonzero node corresponds to that exemplar class which best matches the input pattern.

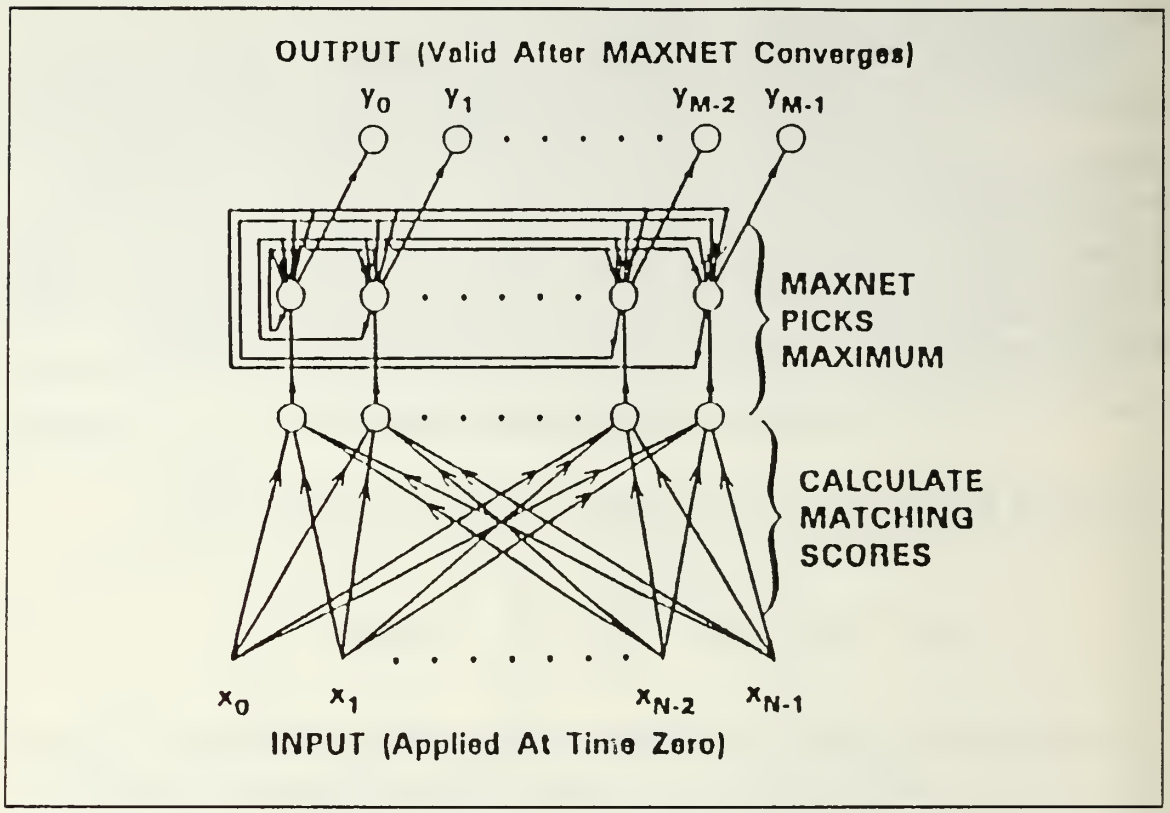


Figure 14. The complete neural network classifier referred to as The Hamming net [Ref. 2]

The block diagram of the complete Hamming net, when used as a classifier, is completed by putting together the feed-forward neural net referred to as the lower subnet (calculates the weighted sums) , and the maxnet referred to as the upper subnet (selects the node with the maximum output value). The complete Hamming net is then as shown in Figure 14.

C. IMPLEMENTATION OF THE HAMMING NET :

The operation algorithm of the Hamming net as a classifier can be described in four steps which the net must follow to classify a certain input pattern. The four steps of the algorithm are : [Ref. 2]

Step 1. Assign connection Weights and Offsets

In the lower subnet :

$$w_{ij} = \frac{x_i^j}{2} \qquad 0 \le i \le N-1\,,\ 0 \le j \le M-1 \tag{3-13}$$

and

$$\theta_j = \frac{N}{2} \qquad 0 \le j \le M-1 \tag{3-14}$$

In the upper subnet :

$$t_{kl} = \begin{cases} +1, & k = l \\ -\varepsilon, & k \neq l, \quad \varepsilon < \dfrac{1}{M}, \quad 0 \le k,l \le M-1 \end{cases} \tag{3-15}$$

In these equations, w_{ij} is the connection weight from input i to node j in the lower subnet and θ is the threshold in that node. The connection weight from node k to node l in the upper subnet is t_{kl} and all thresholds in this subnet are zero. x_i^j is element i of exemplar j .

Step 2. Initialize with Unknown Input Pattern

$$\mu_j(0) = f_t\left(\sum_{i=0}^{N-1} w_{ij}\, x_i - \theta_t\right) \qquad 0 \le j \le M-1 \tag{3-16}$$

In this equation, $\mu_j(t)$ is the output of node j in the upper subnet at time t, x_i is element i of the input, and f_t is the threshold logic nonlinearity. It is assumed that the maximum input to this nonlinearity never causes the output to saturate.

Step 3. Iterate Until Convergence : using Equation 3-10, this process is repeated until convergence occurs after which the output of only one node remains positive.

Step 4. Repeat by Going to Step 2

First, weights and thresholds are set using the N elements of each one of the M stored patterns, as shown in Step 1 of the above algorithm. Then a binary input pattern

with N bits (N = 120 elements in this implementation) is presented at the bottom of the Hamming net at time zero. The N bits of the input pattern were chosen to take on the values +1 and −1 for +1 and −1 states, respectively. This input pattern must be presented long enough to allow the lower subnet of the net to calculate matching scores which are going to be fed to the upper subnet (maxnet) allowing it to settle and initialize its outputs. These matching scores are equal to N minus the Hamming distances between the input and each one of the M exemplar patterns. This operation is done by using the equation given in Step 2 of the previous algorithm. The input is then removed and the maxnet iterates until convergence using the iteration formula 3-10. By convergence, we mean that the output of only one node, corresponding to one of the M stored patterns, is a nonzero value. Classification is then terminated and the nonzero maxnet output node will point out the selected class that best matches the input pattern.

The M stored patterns used in the implementation of the Hamming net when used as a classifier, were choosen to be similar to those used in the previous chapter; however, here we are using 10 exemplar patterns (M = 10) instead of 8 used earlier. The 10 stored patterns (Figures 15 and 16) consist of 120 nodes (N = 120) each as in the previous chapter (12 by 10 representation matrices). These 10 classes were choosen to represent all digits.

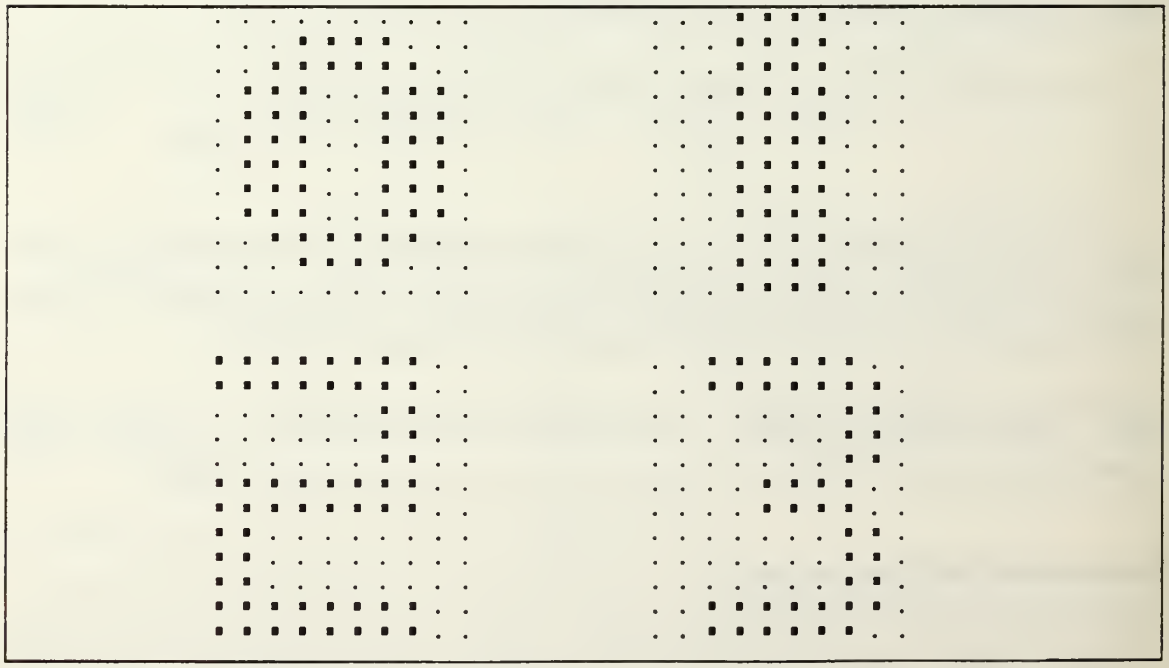


Figure 15. The first four stored patterns

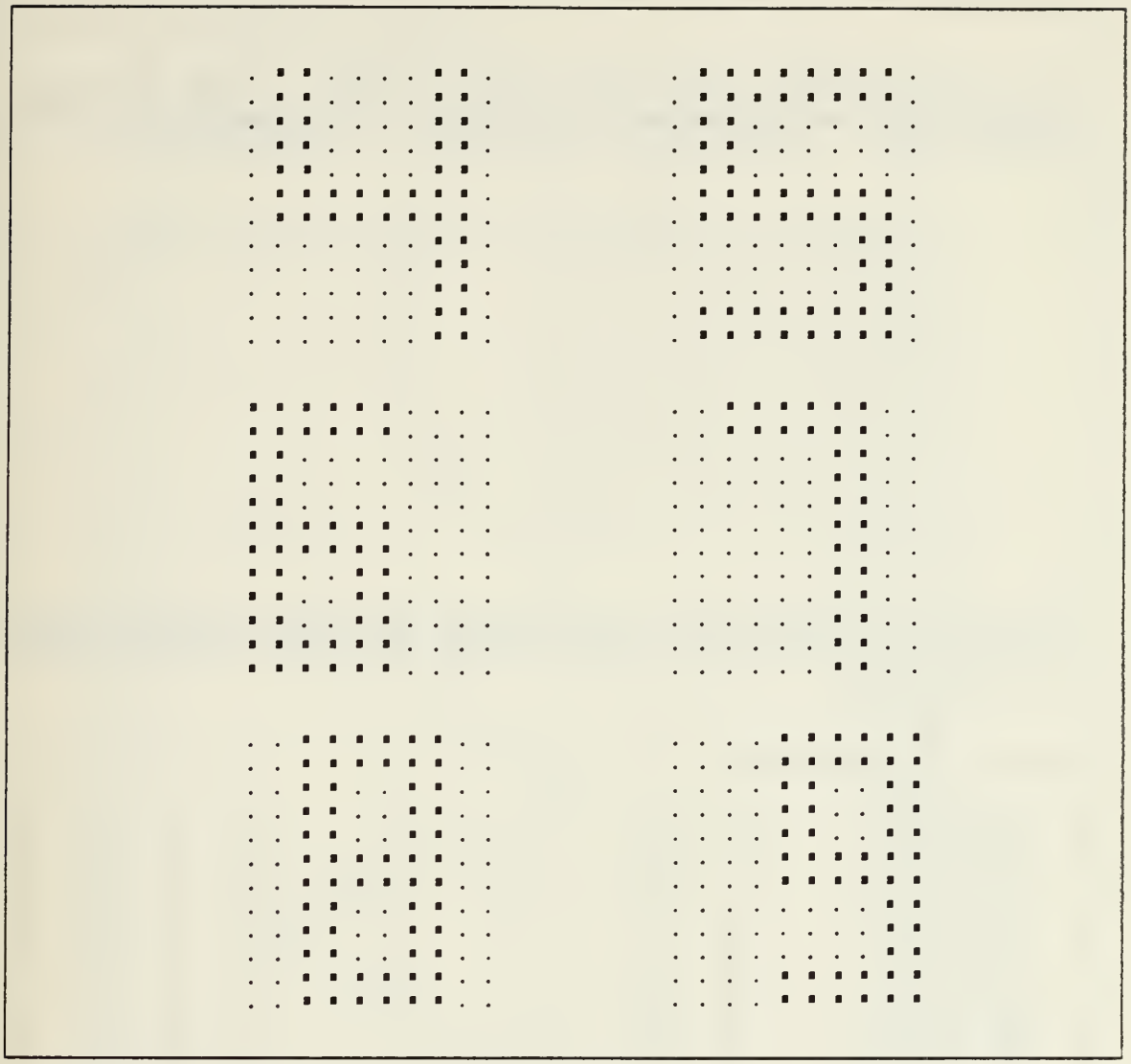


Figure 16. The rest of the 10 stored patterns

D. SIMULATION OF THE HAMMING NETWORK:

Using the Fortran program provided in Appendix B. we simulated the operation of the Hamming net when it is used as a classifier. The input pattern is choosen to be the digit '3' pattern which in the same fashion as for the Hopfield net, reversing its bits randomly from +1 to −1 and vice versa with the same probability (0.25).

The behavior of the Hamming net is illustrated in the output of the simulation program provided in Figure 17.

THE UNKNOWN INPUT PATTERN TO THE HAMMING NETWORK, ON THE LEFT AS IMPOSED ON THE NETWORK AND ON THE RIGHT IN A MUCH CLEAR REPRESENTATION WHERE EVERY (▪) REPLACES A 1 AND EVERY (.) REPLACES A -1:

```
-1 -1 -1  1  1  1  1 -1 -1  1      .  .  .  ▪  ▪  ▪  ▪  .  .  ▪
-1  1 -1  1  1  1  1  1  1  1      .  ▪  .  ▪  ▪  ▪  ▪  ▪  ▪  ▪
 1 -1 -1 -1  1 -1 -1  1 -1 -1      ▪  .  .  .  ▪  .  .  ▪  .  .
 1 -1 -1 -1 -1 -1 -1 -1  1 -1      ▪  .  .  .  .  .  .  .  ▪  .
-1 -1 -1 -1 -1  1 -1 -1  1 -1      .  .  .  .  .  ▪  .  .  ▪  .
-1 -1  1 -1  1  1  1  1 -1 -1      .  .  ▪  .  ▪  ▪  ▪  ▪  .  .
 1 -1 -1 -1  1  1 -1 -1 -1 -1      ▪  .  .  .  ▪  ▪  .  .  .  .
-1 -1 -1  1 -1 -1 -1 -1  1  1      .  .  .  ▪  .  .  .  .  ▪  ▪
-1 -1 -1 -1 -1 -1  1  1  1 -1      .  .  .  .  .  .  ▪  ▪  ▪  .
-1 -1 -1  1 -1 -1 -1  1 -1 -1      .  .  .  ▪  .  .  .  ▪  .  .
 1  1  1  1  1 -1  1 -1 -1 -1      ▪  ▪  ▪  ▪  ▪  .  ▪  .  .  .
-1 -1 -1  1  1  1 -1  1 -1  1      .  .  .  ▪  ▪  ▪  .  ▪  .  ▪
```

THE OUTPUT OF THE HAMMING NETWORK, WHERE EACH COLUNM REPRESENTS THE OUTPUT NODE VALUES FOR THE CORRESPONDING CLASSES AT A CERTAIN NUMBER OF ITERATIONS:

NUMB OF ITERATIONS=	1	2	3	4	5	6	7	8	9	10
FOR CLASS 0:	59	5	0	0	0	0	0	0	0	0
FOR CLASS 1:	73	20	5	0	0	0	0	0	0	0
FOR CLASS 2:	71	18	3	0	0	0	0	0	0	0
FOR CLASS 3:	89	38	25	21	19	18	17	16	16	16
FOR CLASS 4:	63	9	0	0	0	0	0	0	0	0
FOR CLASS 5:	75	22	7	1	0	0	0	0	0	0
FOR CLASS 6:	65	12	0	0	0	0	0	0	0	0
FOR CLASS 7:	75	22	7	1	0	0	0	0	0	0
FOR CLASS 8:	79	27	13	8	5	3	1	0	0	0
FOR CLASS 9:	77	25	11	6	3	1	0	0	0	0

CLASSIFICATION OF THE UNKNOWN INPUT PATTERN:

THEN, THE DISTURBED UNKNOWN INPUT PATTERN TO THE HAMMING NETWORK CORRESPONDS TO THE PATTERN STORED OF THE CLASS THREE.

Figure 17. Response of the Hamming net to the first input pattern

For more clarity, we are going to illustrate the behavior of the Hamming net in the following plots. Each plot corresponds to the output of the net at the corresponding time step from time zero until its convergence.

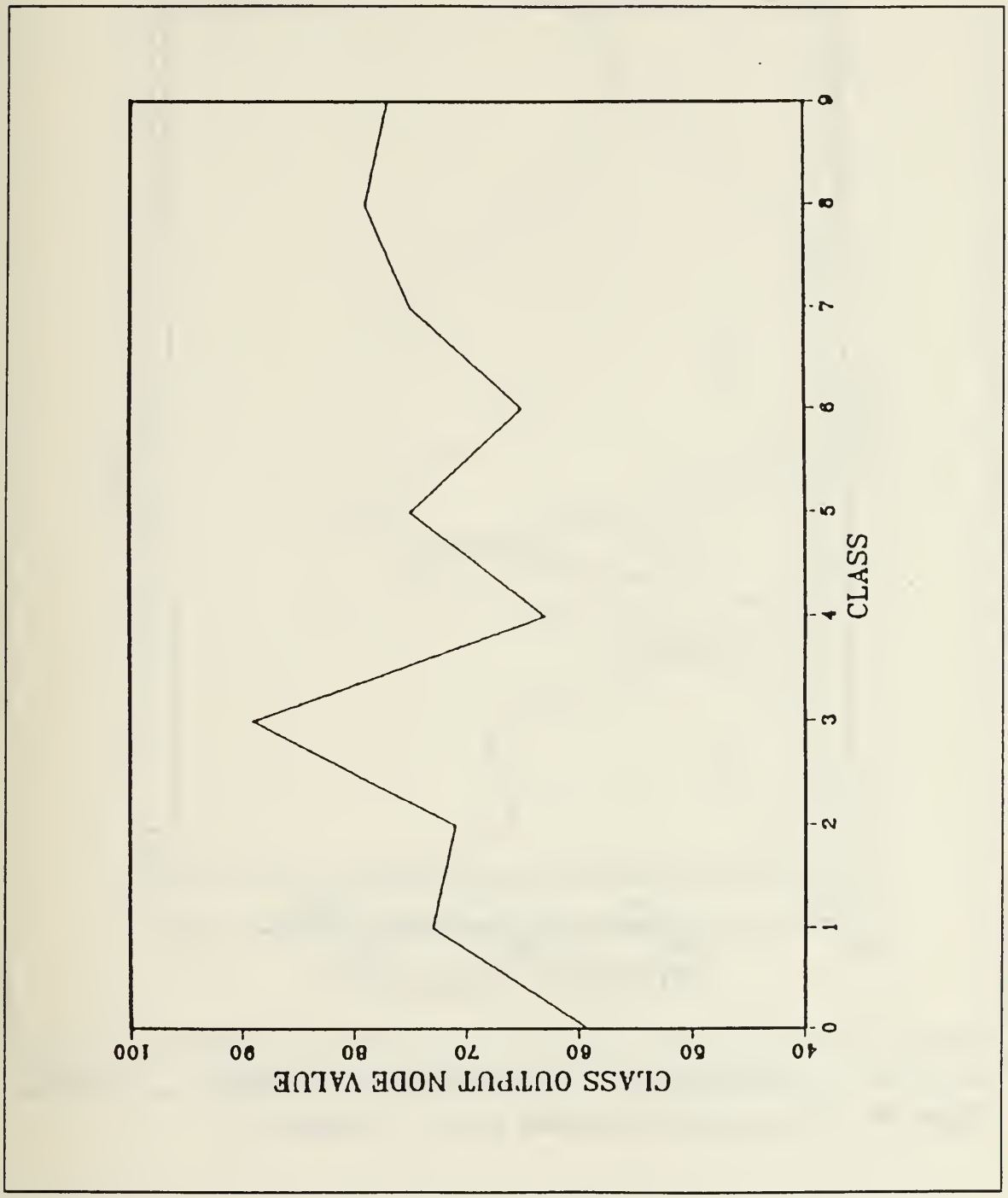


Figure 18. The output of the Hamming net at t = 1 for digit "3"

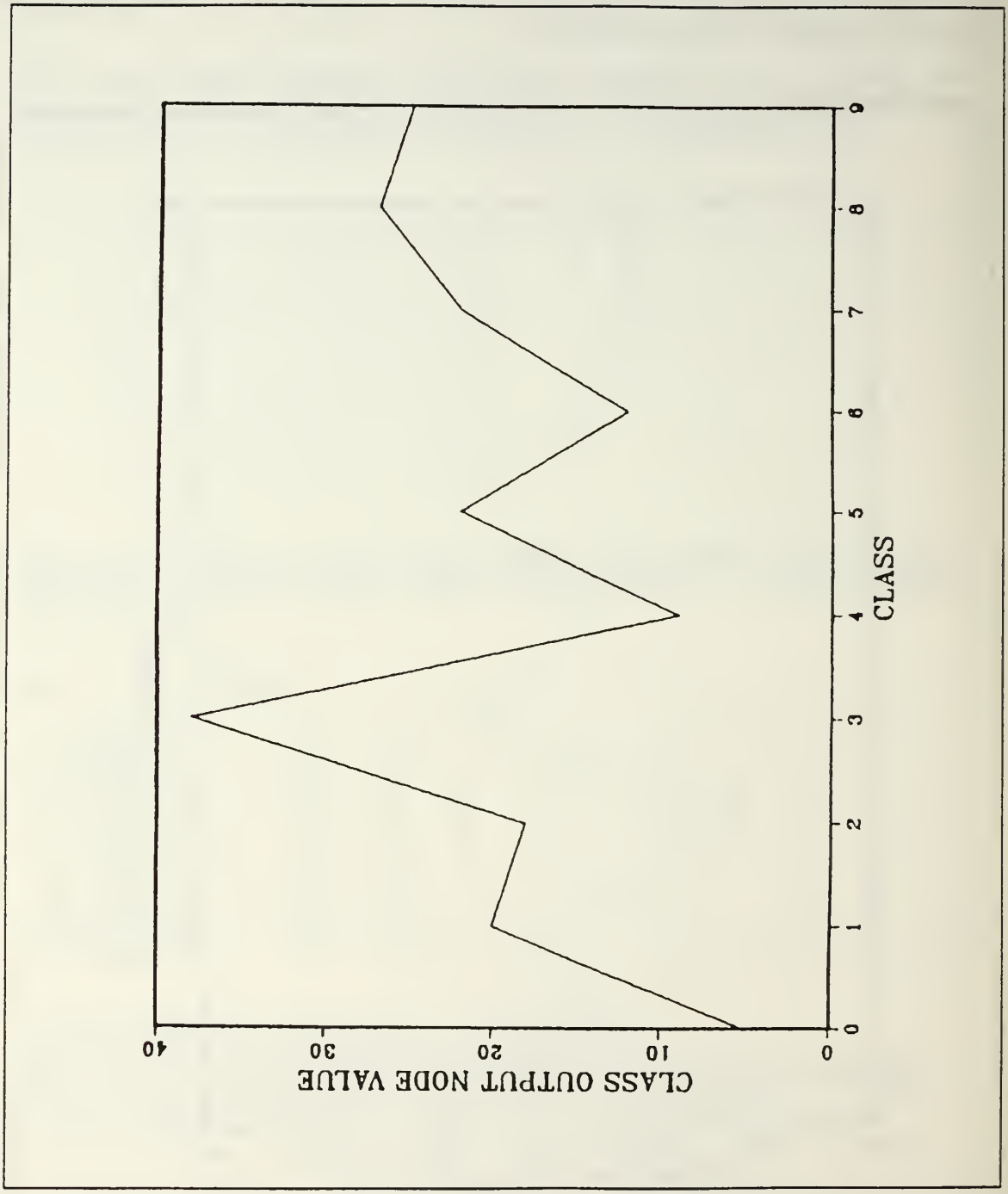


Figure 19. The output of the Hamming net at t = 2 for digit "3"

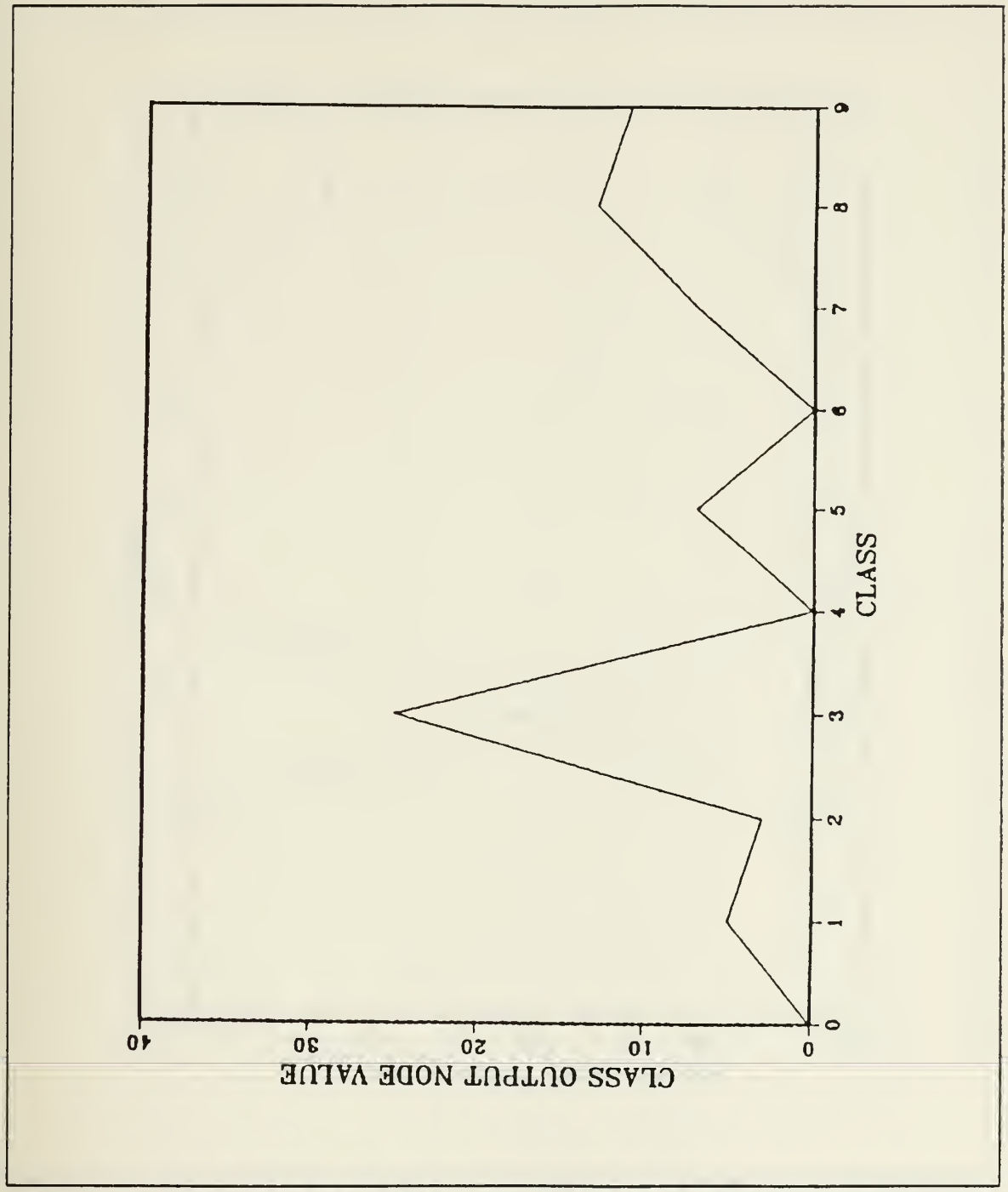


Figure 20. The output of the Hamming net at t = 3 for digit "3"

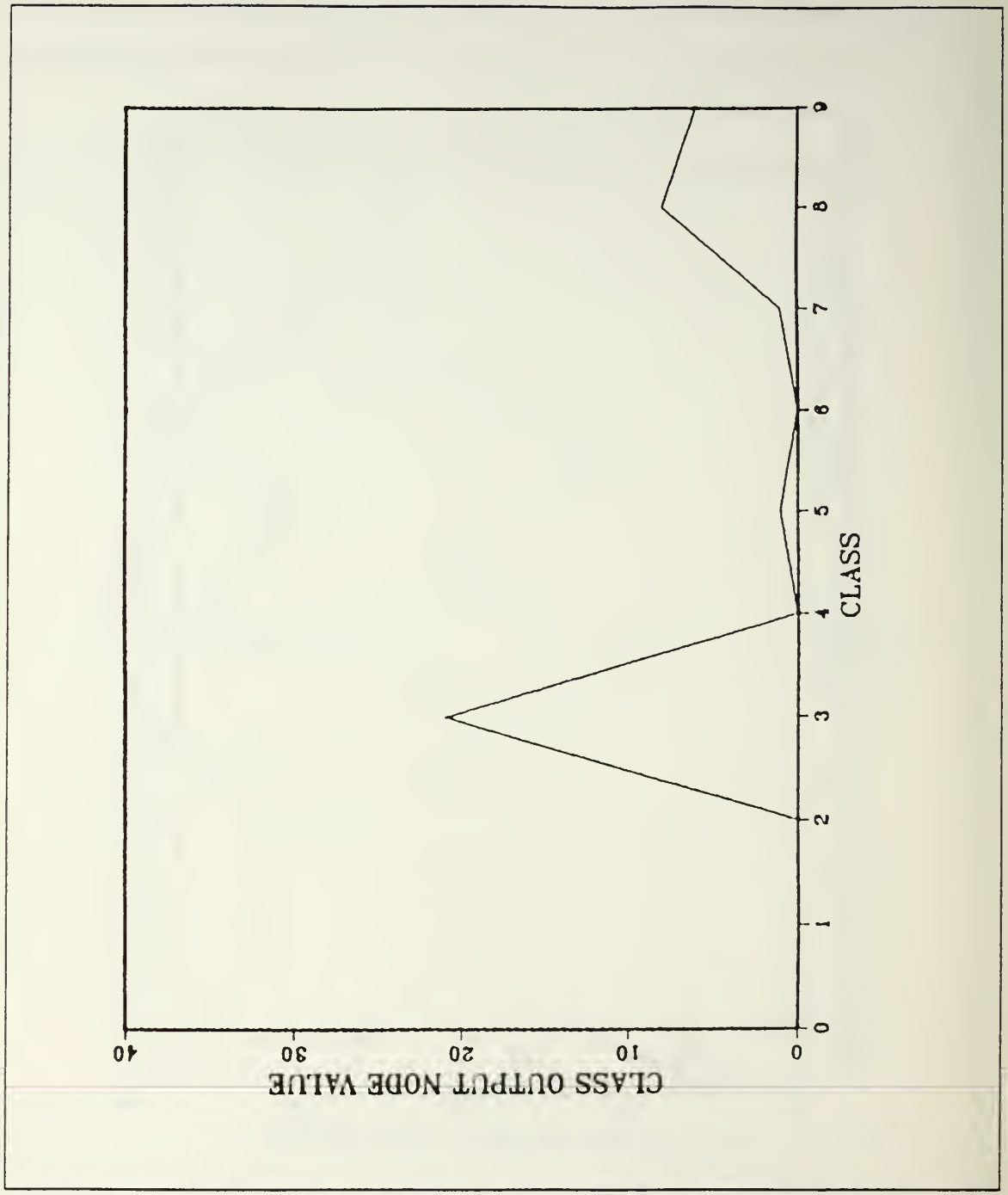


Figure 21. The output of the Hamming net at t = 4 for digit "3"

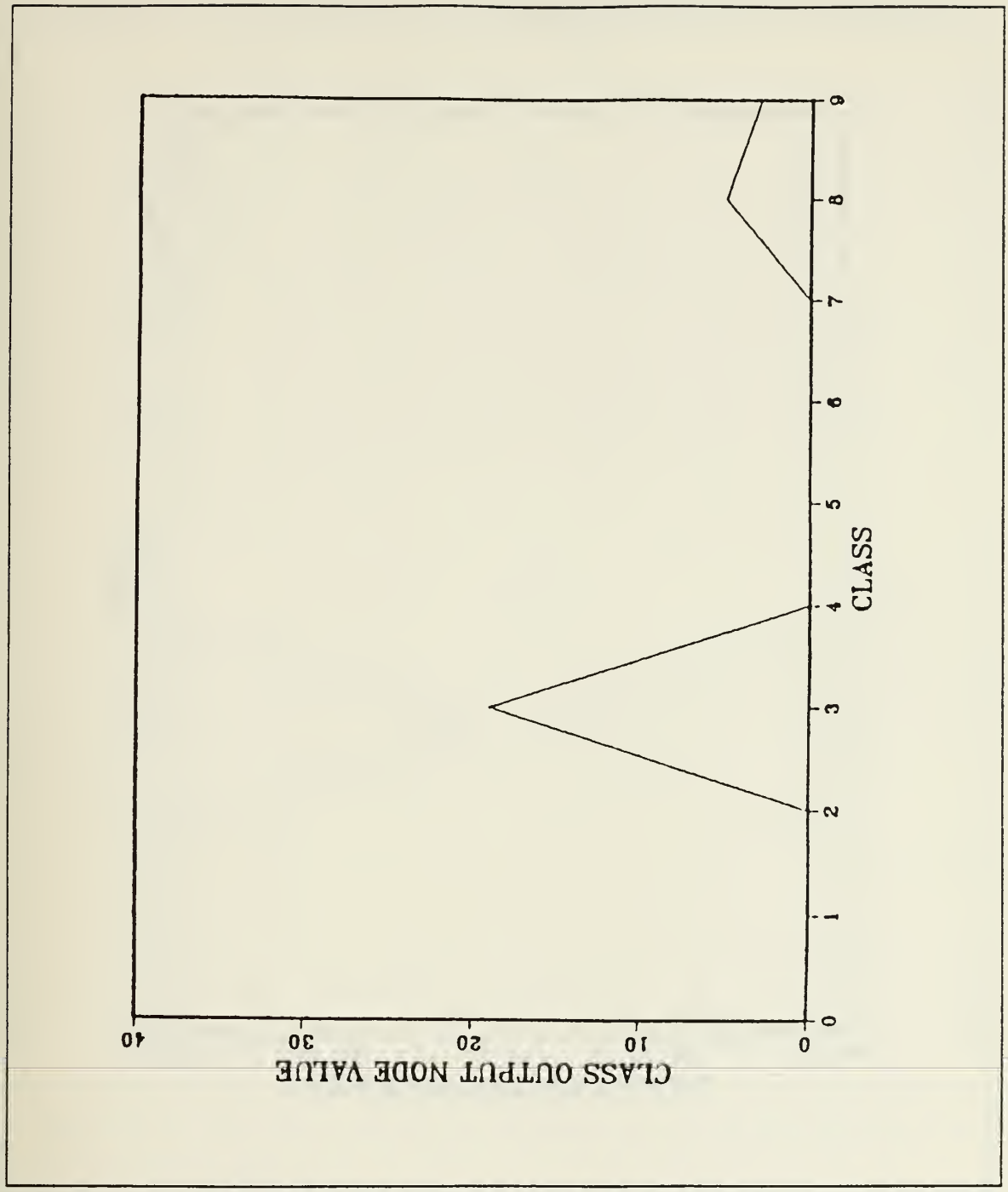


Figure 22. The output of the Hamming net at t = 5 for digit "3"

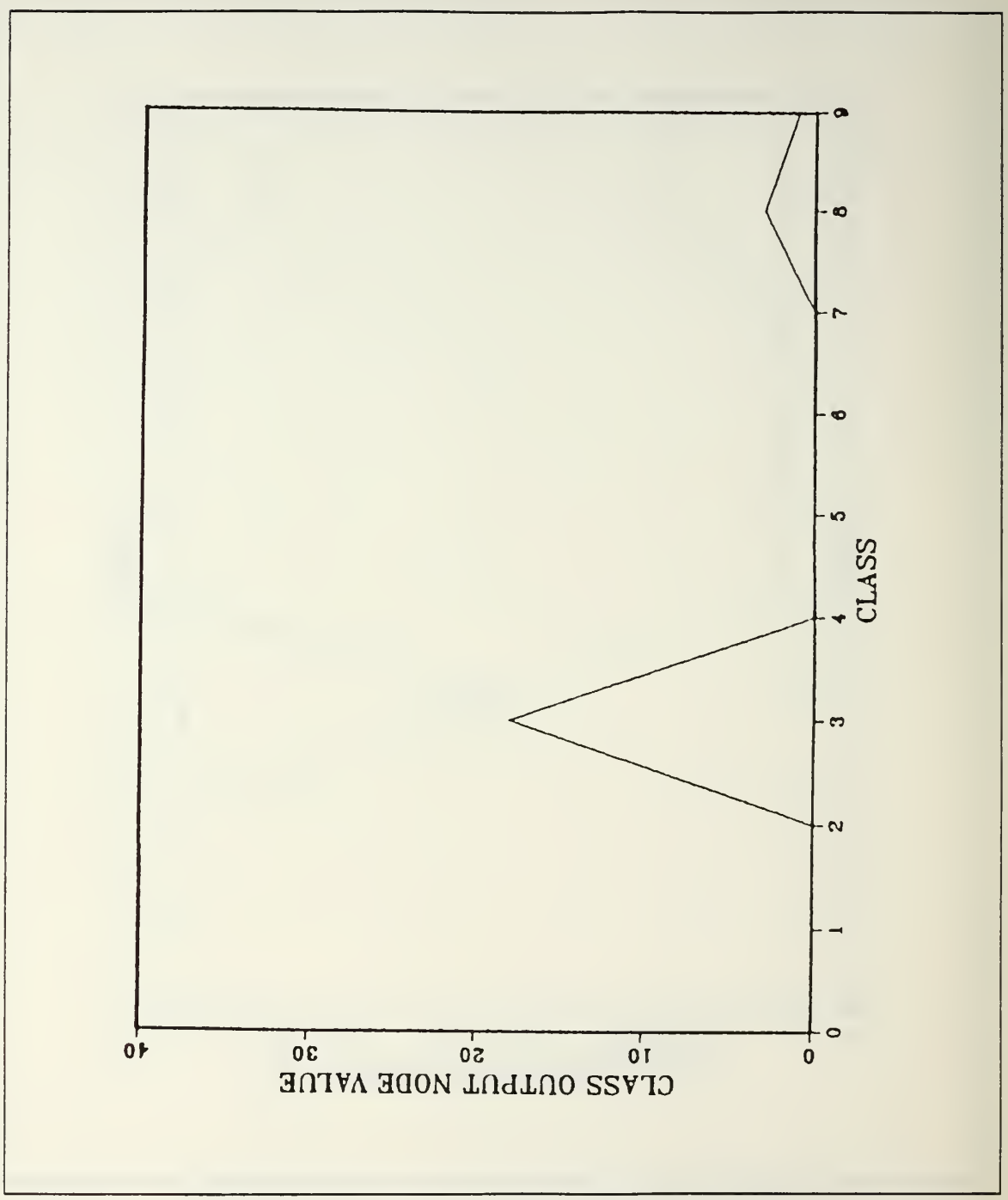


Figure 23. The output of the Hamming net at t = 6 for digit "3"

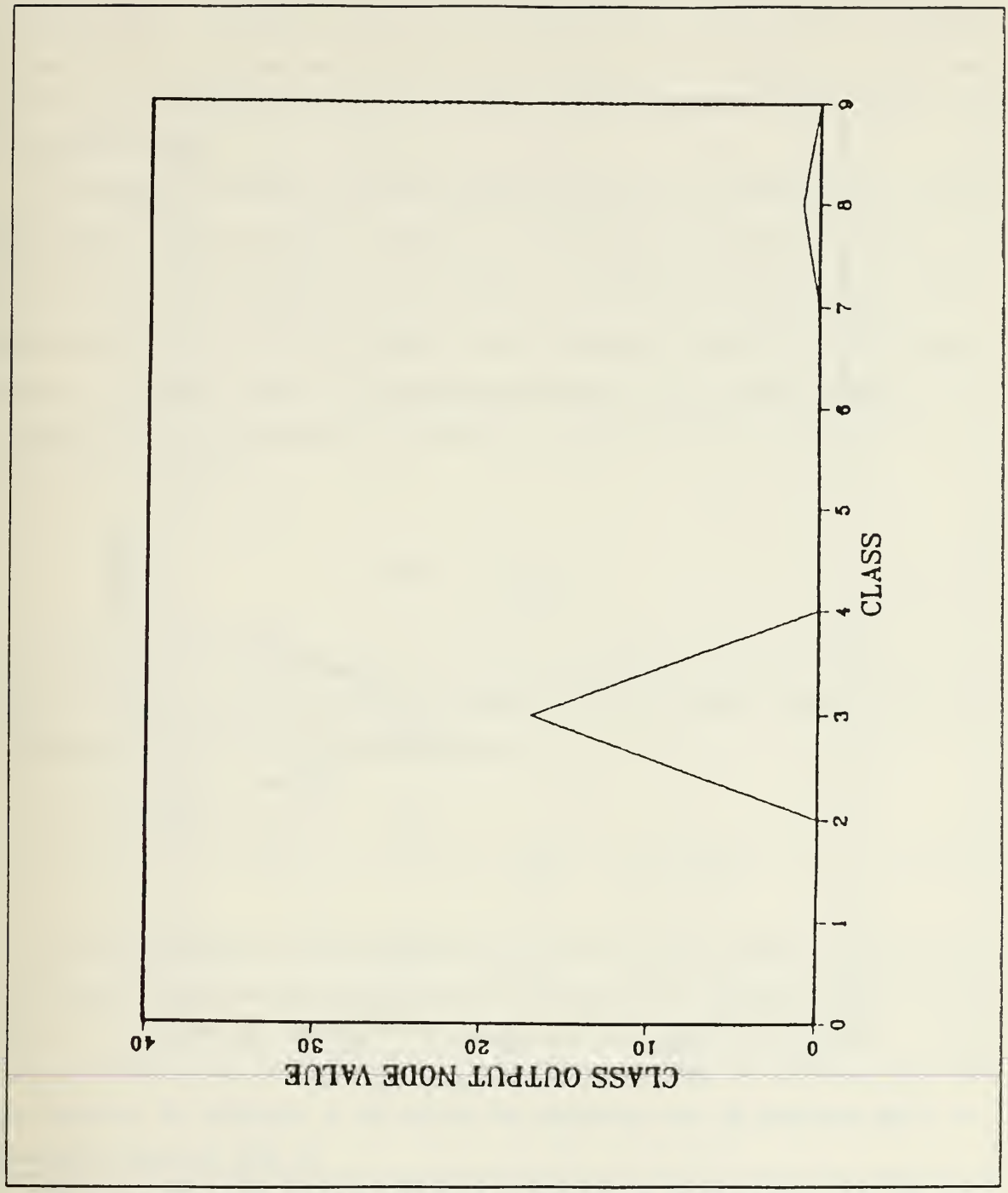


Figure 24. The output of the Hamming net at t = 7 for digit "3"

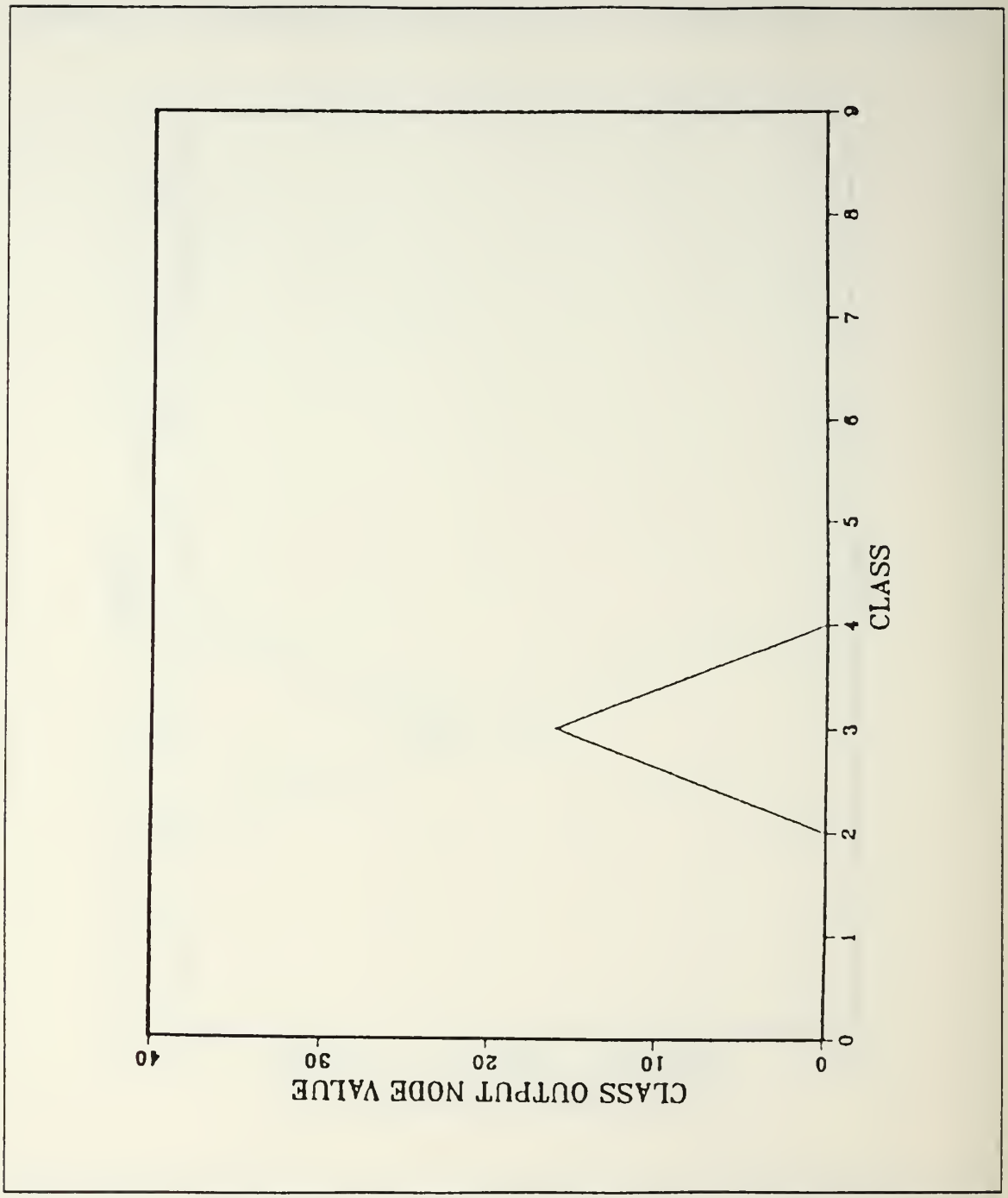


Figure 25. The output of the Hamming net at t = 8 for digit "3"

As you can see, the net has effectively converged to the correct class corresponding to digit '3' and only 8 iterations were required to do so. By convergence, we mean that

the output nodes stop changing in time steps, and only the output of one node is a nonzero value. In our case, it was the output node of class 3. We say that the net has converged to the class that best matches the input pattern which is given by the position of the nonzero output node within the others. Further iterations will not change the output node values.

The proof of convergence depends primarily on the fact the inhibition to the node containing the maximum value, in our case it was the node corresponding to class '3', is always less than the inhibition to other nodes. This explains the fact that all output node values were decreasing on successive time steps. The output node values of those classes that are close to the input pattern were decreasing in time, but not in the same fashion as the classes that are totally different from the input pattern which were decreasing faster. At convergence, the inhibition to the node with the maximum value reduces to zero. [Ref. 4]

$$inhib_j(t) = \sum_{i \neq j} t_{ij}\, \mu_i(t) \qquad (3-17)$$

where, $inhib_j(t)$ is the second term of the right hand side of Equation 3-10, $\mu_i(t)$ is the output of node i at time t and t_{ij} is the inhibition weight between nodes in the upper subnet and it's given by the following formula,

$$t_{ij} = \begin{cases} +1, & i = j \\ -\varepsilon, & i \neq j, \quad \varepsilon < \dfrac{1}{M}, \quad 0 \leq i,j \leq M-1 \end{cases} \qquad (3-18)$$

Node 3 corresponds to the maximum input, then on the first iteration, the inhibition to this node was less than the inhibition to all other nodes. This follows because all node outputs are positive and the sum of all outputs, excluding one in Equation 3-17, will be minimum when the maximum is excluded. Node 3 thus remains the maximum after the first iteration. By induction, it will remain the maximum over all iterations and it responded as expected. [Ref. 4]

The remainder of the proof of convergence depends on demonstrating that the output of node 3 is never driven to zero, but the outputs of all other nodes are. When Equation 3-12 is satisfied, $inhib_j(t)$ is always less than the average value of all other node outputs. The inhibition to node 3 will thus be less than the average of the output of all nodes. Whenever a maximum exists, this inhibition will always be less than the current

output of node 3 because the maximum of a set of positive numbers is always greater than the average. The output of node 3 will not be driven to zero while any other nodes have nonzero outputs. After all other node outputs are driven to zero, the inhibition to node 3 drops to zero, and the output of node 3 remains constant. The output of all other nodes will always be driven to zero because the inhibition to these nodes remains positive on all iterations and approaches a positive constant as time increases. In practice, the maxnet will still converge and find the maximum when each weight w_{ij} is set to $\frac{1}{M-1}$ plus a small random component. This forces the net to find a maximum when the inputs to all nodes are identical. As a matter of fact, this discussion can be generalized to all input patterns. [Ref. 4]

Then, we presented another input pattern which is of the digit "9". This pattern was noise corrupted in similar fashion as for digit "3". The purpose of this simulation was to see if the net will behave as discussed earlier and if the number of iterations necessary for a successful convergence is function of the input pattern and how much noise disturbed it is. So, for the noisy pattern of digit "9", the response of the net is as provided in Figure 26.

For more clarity, we are going to illustrate the response of the net in the following graphs for successive iterations. Each plot corresponds to the response of the net at the corresponding number of iterations, where the class output node values are decreasing from an iteration to the next. By inhibition, all of them will be driven to zero, some faster than others, except for the class output node corresponding to that to the stored exemplar that best matches the unknown input pattern. Then, this output node value will remain constant throughout future iterations while the zero valued output nodes will remain at zero.

THE UNKNOWN INPUT PATTERN TO THE HAMMING NETWORK, ON THE LEFT AS IMPOSED ON THE NETWORK AND ON THE RIGHT IN A MUCH CLEAR REPRESENTATION WHERE EVERY (▪) REPLACES A 1 AND EVERY (.) REPLACES A -1:

```
-1  1  1 -1  1  1  1 -1  1 -1      .  ▪  ▪  .  ▪  ▪  ▪  .  ▪  .
-1 -1 -1  1  1  1 -1  1  1  1      .  .  .  ▪  ▪  ▪  .  ▪  ▪  ▪
 1  1 -1 -1 -1 -1 -1 -1 -1 -1      ▪  ▪  .  .  .  .  .  .  .  .
-1 -1 -1 -1  1  1 -1  1  1  1      .  .  .  .  ▪  ▪  .  ▪  ▪  ▪
-1 -1  1 -1  1  1 -1 -1  1 -1      .  .  ▪  .  ▪  ▪  .  .  ▪  .
-1 -1 -1  1 -1  1  1  1  1  1      .  .  .  ▪  .  ▪  ▪  ▪  ▪  ▪
 1  1 -1 -1  1 -1 -1  1  1 -1      ▪  ▪  .  .  ▪  .  .  ▪  ▪  .
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1      .  .  .  .  .  .  .  .  .  .
-1 -1 -1  1 -1 -1 -1 -1  1  1      .  .  .  ▪  .  .  .  .  ▪  ▪
 1  1  1 -1 -1 -1 -1 -1  1  1      ▪  ▪  ▪  .  .  .  .  .  ▪  ▪
-1 -1 -1 -1 -1  1 -1  1  1  1      .  .  .  .  .  ▪  .  ▪  ▪  ▪
-1 -1 -1 -1  1  1  1  1  1  1      .  .  .  .  ▪  ▪  ▪  ▪  ▪  ▪
```

THE OUTPUT OF THE HAMMING NETWORK, WHERE EACH COLUNM REPRESENTS THE OUTPUT NODE VALUES FOR THE CORRESPONDING CLASSES AT A CERTAIN NUMBER OF ITERATIONS:

NUMB. OF ITERATIONS=	1	2	3	4	5	6	7	8	9	10
FOR CLASS 0:	48	0	0	0	0	0	0	0	0	0
FOR CLASS 1:	58	10	0	0	0	0	0	0	0	0
FOR CLASS 2:	60	12	0	0	0	0	0	0	0	0
FOR CLASS 3:	74	27	15	9	5	2	0	0	0	0
FOR CLASS 4:	72	25	12	6	2	0	0	0	0	0
FOR CLASS 5:	72	25	12	6	2	0	0	0	0	0
FOR CLASS 6:	52	4	0	0	0	0	0	0	0	0
FOR CLASS 7:	66	19	6	0	0	0	0	0	0	0
FOR CLASS 8:	58	10	0	0	0	0	0	0	0	0
FOR CLASS 9:	90	45	34	30	28	27	26	26	26	26

CLASSIFICATION OF THE UNKNOWN INPUT PATTERN:

THEN, THE DISTURBED UNKNOWN INPUT PATTERN TO THE HAMMING NETWORK CORRESPONDS TO THE PATTERN STORED OF THE CLASS NINE.

Figure 26. Response of the Hamming net to the second input pattern

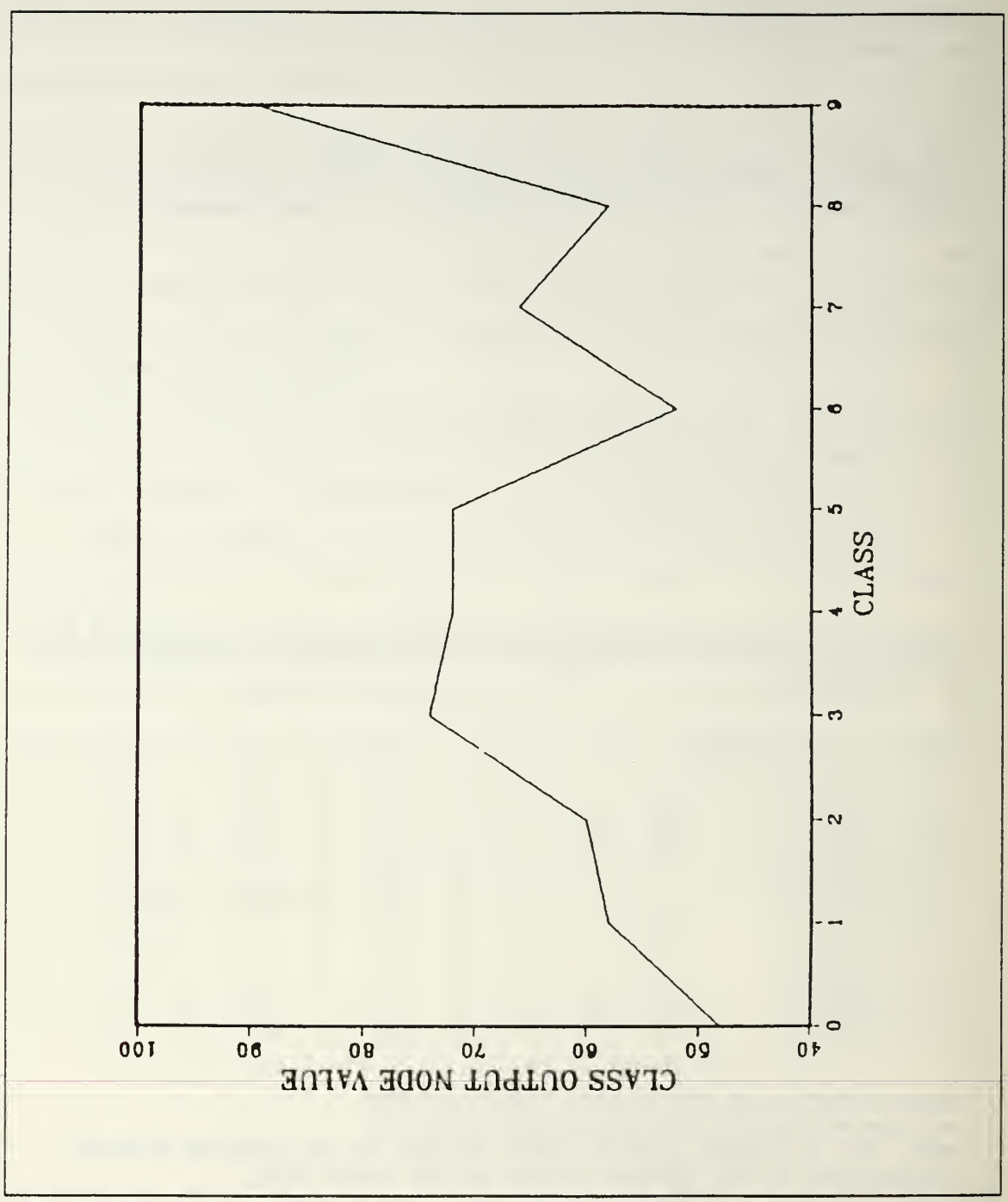


Figure 27. The output of the Hamming net at t = 1 for digit "9"

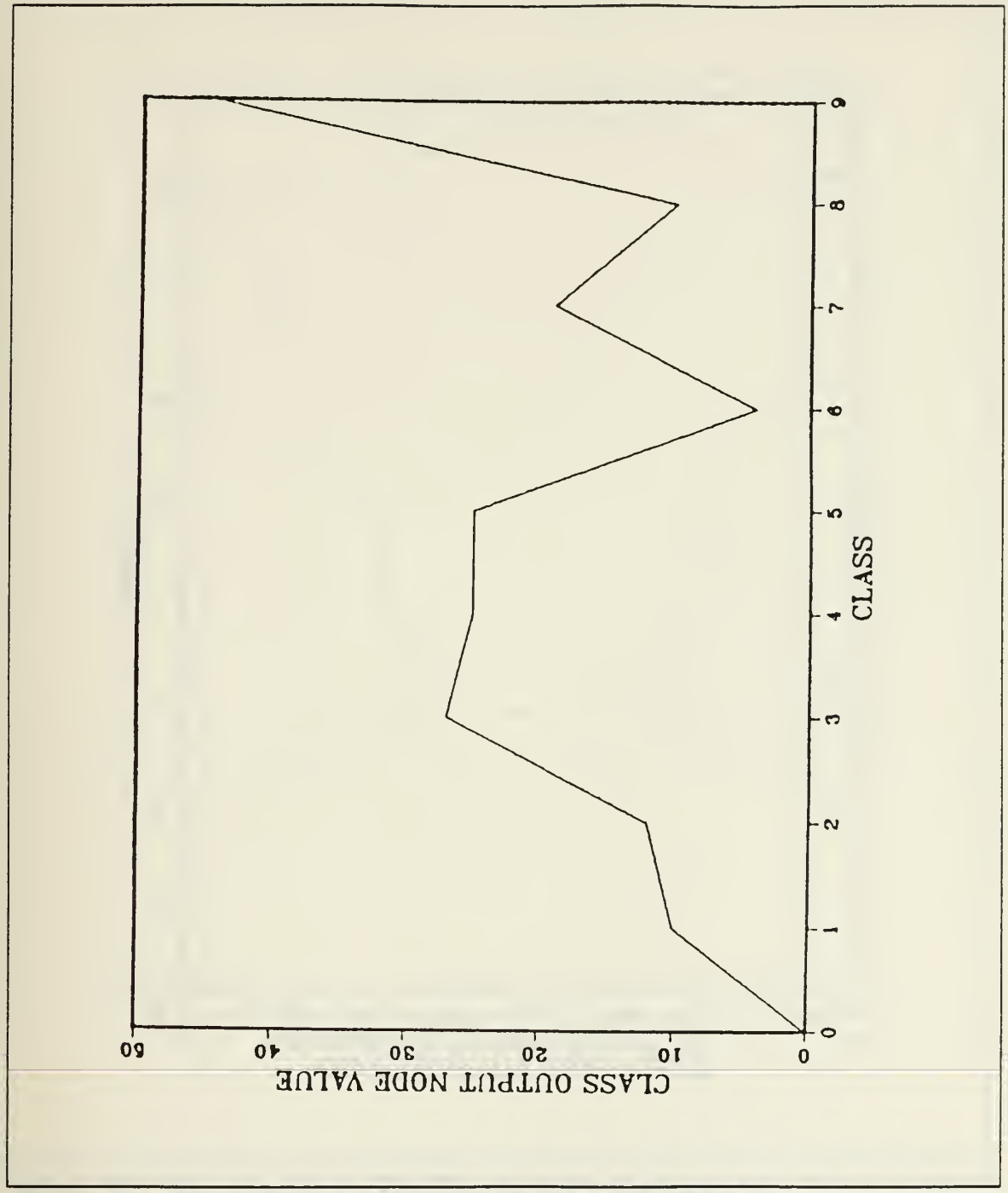


Figure 28. The output of the Hamming net at t = 2 for digit "9"

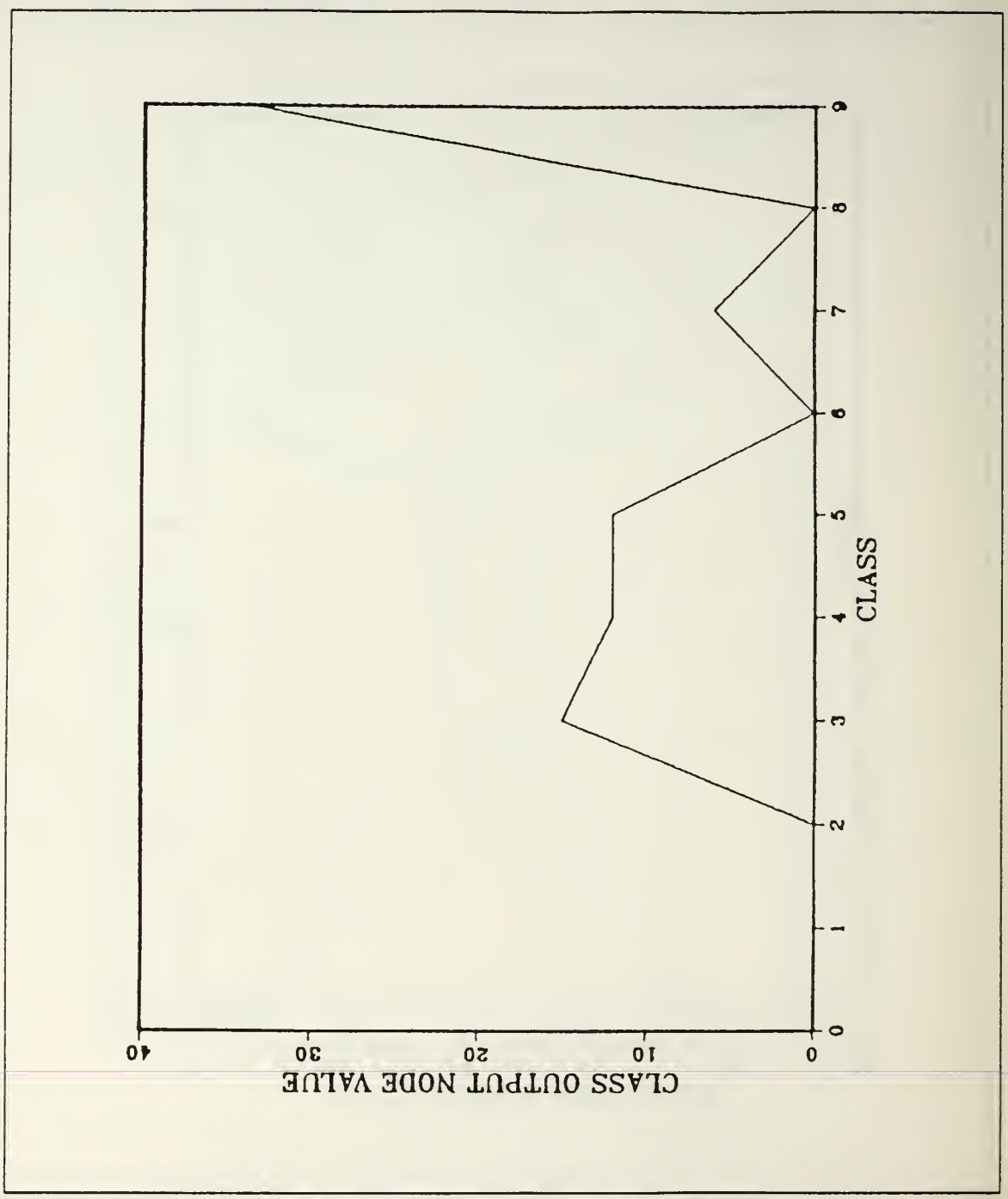


Figure 29. The output of the Hamming net at t = 3 for digit "9"

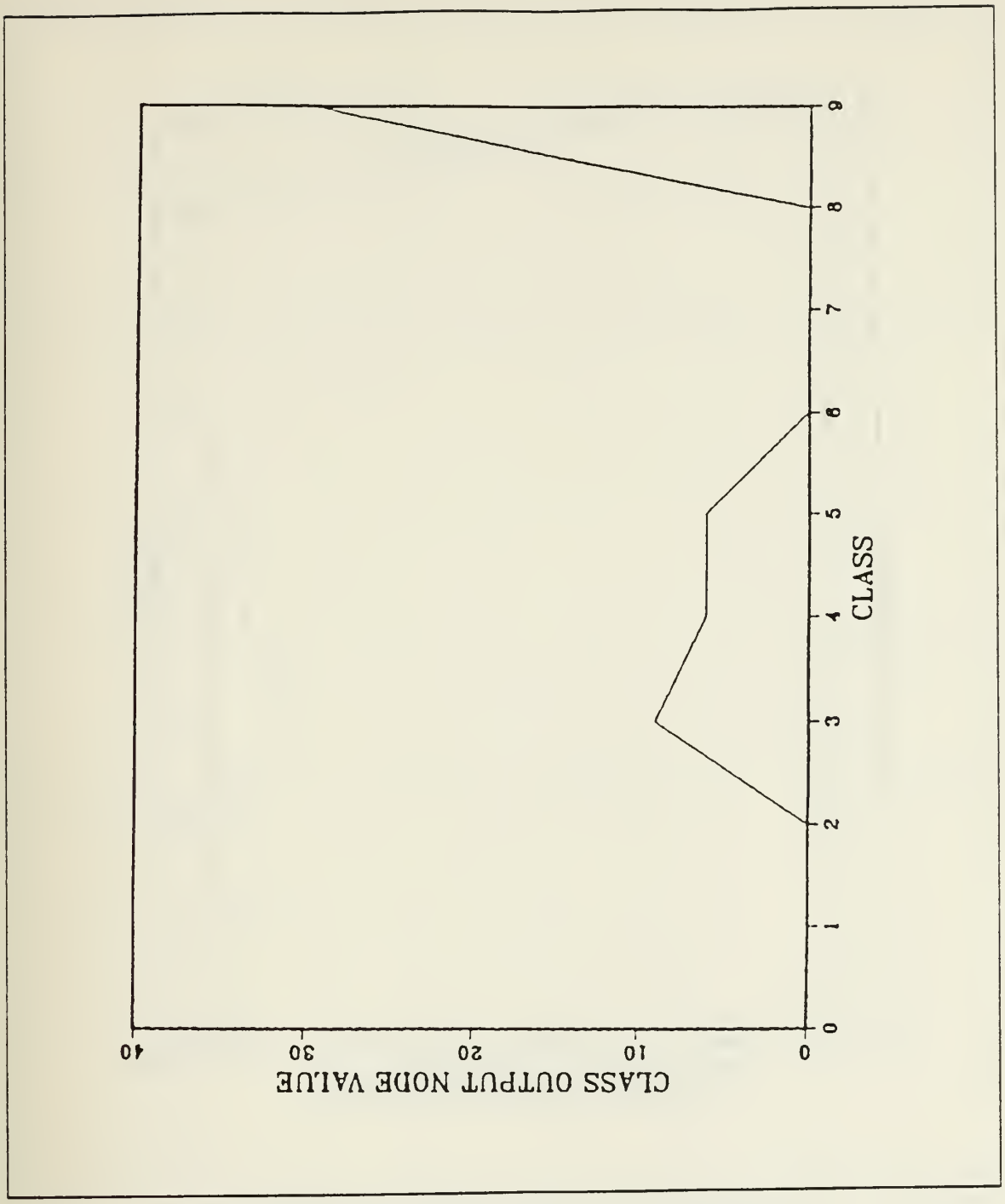


Figure 30. The output of the Hamming net at t = 4 for digit "9"

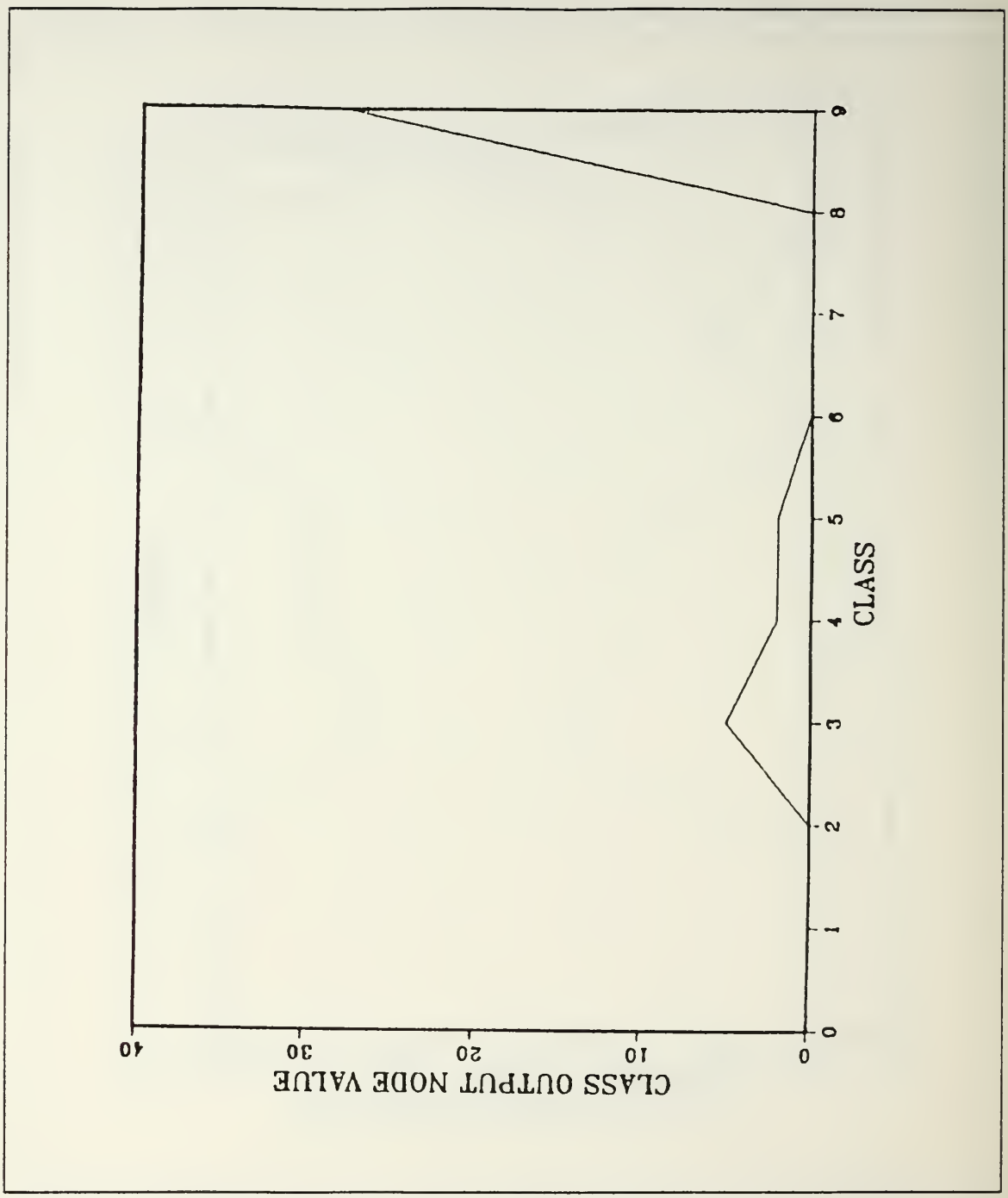


Figure 31. The output of the Hamming net at t = 5 for digit "9"

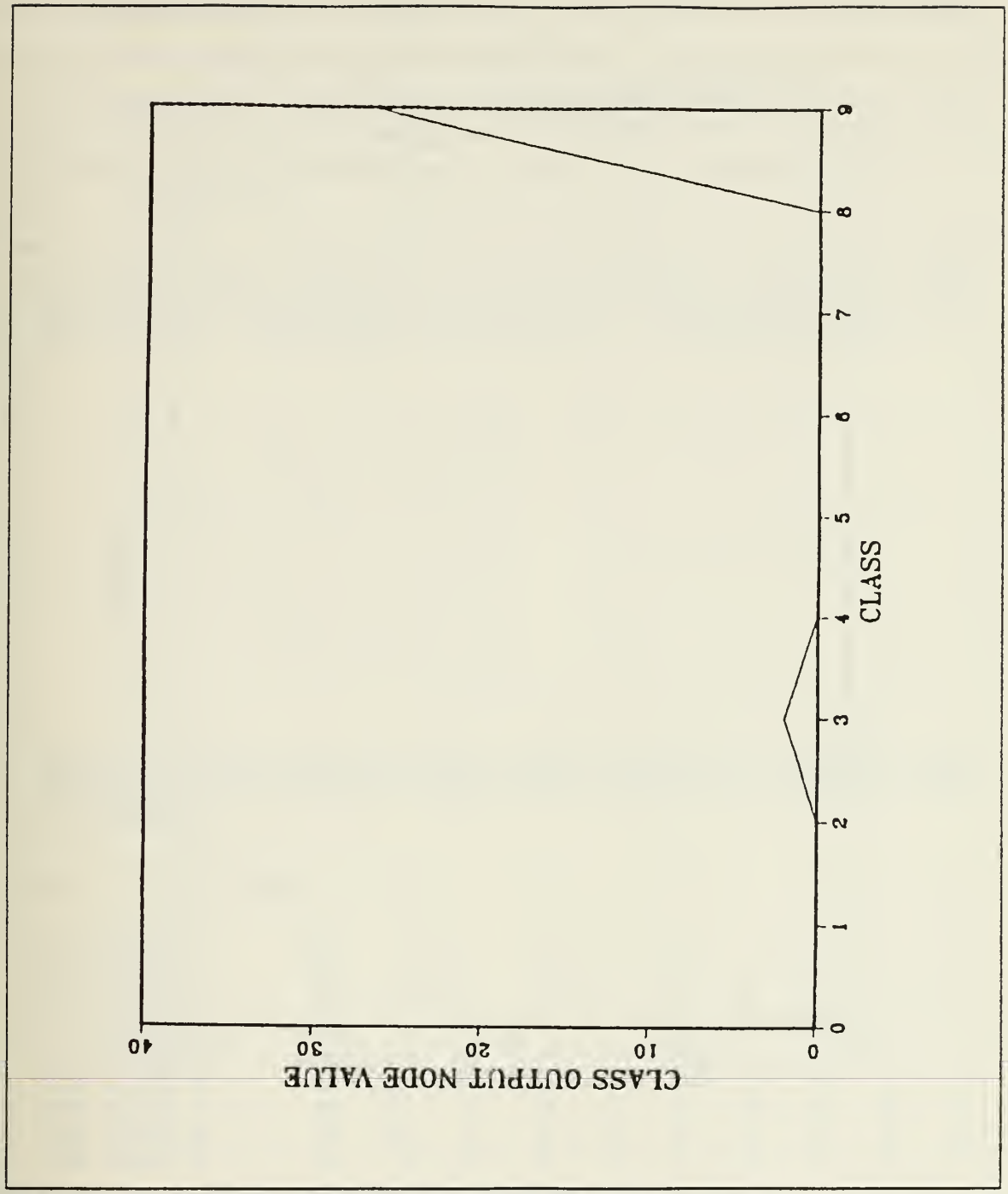


Figure 32. The output of the Hamming net at t = 6 for digit "9"

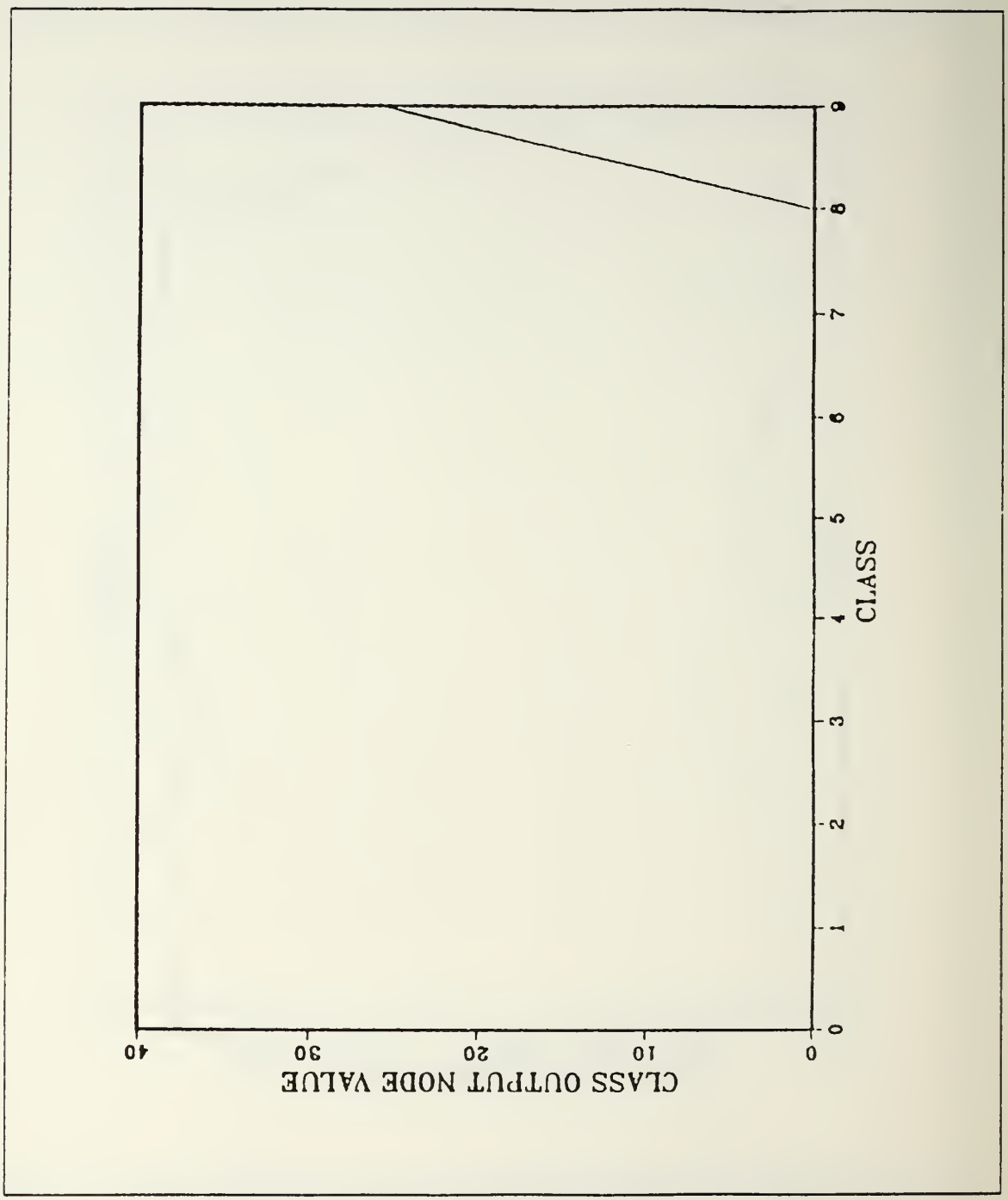


Figure 33. The output of the Hamming net at t = 7 for digit "9"

These graphs have proven once more, the net's mechanism of convergence described earlier. After 7 iterations, the net has effectively converged to the correct pattern of digit

"9", which output node was the only nonzero node while all the others were driven to zero by the convergence process of the Hamming net. Then, seven iterations were used to converge to digit "9" in comparison to eight to recognize digit "3". These simulations were for noise-corrupted input patterns, now we resimulated the net but this time using the perfect pattern of digit "2". The response of the net is provided in Figure 34.

```
THE UNKNOWN INPUT PATTERN TO THE HAMMING NETWORK, ON THE LEFT AS IM-
POSED ON THE NETWORK AND ON THE RIGHT IN A MUCH CLEAR REPRESENTATION
WHERE EVERY ( ▪ ) REPLACES A 1 AND EVERY ( . ) REPLACES A -1:

  1  1  1  1  1  1  1  1 -1 -1      ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ . .
  1  1  1  1  1  1  1  1 -1 -1      ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ . .
 -1 -1 -1 -1 -1 -1  1  1 -1 -1      . . . . . . ▪ ▪ . .
 -1 -1 -1 -1 -1 -1  1  1 -1 -1      . . . . . . ▪ ▪ . .
 -1 -1 -1 -1 -1 -1  1  1 -1 -1      . . . . . . ▪ ▪ . .
  1  1  1  1  1  1  1  1 -1 -1      ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ . .
  1  1  1  1  1  1  1  1 -1 -1      ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ . .
  1  1 -1 -1 -1 -1 -1 -1 -1 -1      ▪ ▪ . . . . . . . .
  1  1 -1 -1 -1 -1 -1 -1 -1 -1      ▪ ▪ . . . . . . . .
  1  1 -1 -1 -1 -1 -1 -1 -1 -1      ▪ ▪ . . . . . . . .
  1  1  1  1  1  1  1  1 -1 -1      ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ . .
  1  1  1  1  1  1  1  1 -1 -1      ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ . .

THE OUTPUT OF THE HAMMING NETWORK, WHERE EACH COLUNM REPRESENTS THE
OUTPUT NODE VALUES FOR THE CORRESPONDING CLASSES AT A CERTAIN NUMBER
OF ITERATIONS:

NUMB. OF ITERATIONS= 1    2    3    4    5    6    7    8    9   10

 FOR CLASS 0:        56    0    0    0    0    0    0    0    0    0
 FOR CLASS 1:        66    9    0    0    0    0    0    0    0    0
 FOR CLASS 2:       120   67   54   49   47   46   45   45   45   45
 FOR CLASS 3:        84   28   12    4    0    0    0    0    0    0
 FOR CLASS 4:        64    7    0    0    0    0    0    0    0    0
 FOR CLASS 5:        84   28   12    4    0    0    0    0    0    0
 FOR CLASS 6:        90   35   20   12    7    3    0    0    0    0
 FOR CLASS 7:        80   24    8    0    0    0    0    0    0    0
 FOR CLASS 8:        78   22    6    0    0    0    0    0    0    0
 FOR CLASS 9:        54    0    0    0    0    0    0    0    0    0

CLASSIFICATION OF THE UNKNOWN INPUT PATTERN:
============================================
THEN, THE DISTURBED UNKNOWN INPUT PATTERN TO THE HAMMING NETWORK
CORRESPONDS TO THE PATTERN STORED OF THE CLASS TWO.
```

Figure 34. Response of the Hamming net to the perfect input pattern

Even for the perfect input pattern of digit "2", the net took 7 iterations to successfully converge to the correct response. So, the number of iterations is definitly not function of the input pattern and how noise disturbed it is but with how many stored patterns it shares many similarities (almost same distribution of +1 and −1). The number of iterations necessary for a successful convergence depends on the output node values at the first iteration. The bigger magnitude these values have, the more similar their respective patterns are to the input and the more iterations will be necessary to drive them to zero by inhibition except for the correct output node.

In our last simulation with digit "2" as a perfect input pattern, after the fourth iteration, only those output nodes corresponding to class "2" and class "6" were the only nonzero valued nodes. We can conclude, from this, that the input pattern, after the fourth iteration, is very close to the patterns of digit "6" and "2", but more closer to "2" than "6" because of its higher output node value. The network took two more iterations to drive the output node of the "6" to zero. Then, the inhibition to node "2" drops to zero and the value of its output node remains constant. The input pattern is said to be digit "2" pattern.

This discussion can be generalized to explain the previous behavior of the network to digit "3" and digit "9" as input patterns. In all cases, the output node with the higher magnitude, after the first iteration is completed, is always the correct node but the convergence mechanism of the net is not to make decisions at this stage.

IV. THE CARPENTER / GROSSBERG NET

A. GENERALITIES :

Classified as a self-organizing neural net, the Carpenter / Grossberg net self-organizes and self-stabilizes its recognition codes in response to arbitrary sequences of binary input patterns. Top-down attentional and matching mechanisms are critical in self-stabilizing the code learning process. The architecture embodies a parallel search scheme which updates itself adaptively as the learning process proceeds. After the learning process has self-stabilized, the search process is automatically disengaged. Thereafter, input patterns directly access their recognition codes without any search. Thus, recognition time does not grow as a function of code complexity.

A novel input pattern can directly access a category if it shares invariant properties with the set of familiar exemplars of that category. These invariant properties emerge in the form of learned critical feature patterns, or prototypes. The architecture possesses a context-sensitive self-scaling property which enables its emergent critical feature patterns to form. They detect and remember statistically predictive configurations of featural elements which are derived from the set of all input patterns that are experienced. Four types of attentional processes (priming, gain control, vigilance, and intermodal competition) are mechanistically characterized. Top-down priming and gain control are needed for code matching and self-stabilization. Attentional vigilance determines how good the learned categories will be. If vigilance increases due to an environmental disconfirmation, then the system automatically searches for and learns the best recognition categories. [Ref. 6]

This chapter develops a theory of how recognition codes are self-organized by a class of neural networks whose qualitative features have been used to analyse data about speech perception, word recognition and recall, visual perception, olfactory coding, evoked potentials, thalamocortical interactions, attentional modulation of critical termination, and amnesia. These networks comprise the adaptive resonance theory (ART) characterized as a system of ordinary differential equations.

B. IMPLEMENTATION OF THE CARPENTER / GROSSBERG NET :

The neural network that will be discussed in this chapter is known as an ART system, after the adaptive resonance theory introduced by Grossberg [Ref. 7], see Appendix C. Recently, ART networks have been further studied and their dynamic properties

have been derived in a series of theorems. These theorems predict both the order of search, as a function of the learning history of the net, and the asymptotic category structure self-organized by an arbitrary binary input sequence.

The operation of the ART system discussed in Appendix C will be used to develop a neural net known as the Carpenter / Grossberg net, using neural net components, which will form clusters and is trained without supervision. The net can learn from input patterns and later differentiate between new and stored (learned) patterns. If the new and unknown input pattern is classified as a previously learned pattern at a certain level of vigilance, it will be ignored, but if it is not, it will be added as a new pattern by the net. This process is repeated for all input patterns. The number of learned patterns thus grows with time and depends strongly on the level of vigilance (threshold) used to compare input patterns to the already stored ones.

The operation of the Carpenter / Grossberg net which forms clusters (learned patterns) and is trained without supervision is given in eight steps : [Ref. 2]

Step 1. Initialization

$$t_{ij}(0) = 1 \qquad (4-1)$$

$$b_{ij}(0) = \frac{1}{1+N} \qquad (4-2)$$

$$0 \le i \le N-1 \quad , \; 0 \le j \le M-1$$

Set ρ , $0 \le \rho \le 1$.

In these equations $b_{ij}(t)$ is the bottom-up and $t_{ij}(t)$ is the top-down connection weight between input node i and output node j at time t as shown in Figure 35. These weights define the exemplar specified by output node j. The fraction ρ is the vigilance threshold which indicates how close an input must be to a stored exemplar to match.

Step 2. Apply New Input

Step 3. Compute Matching Scores

$$\mu_j = \sum_{i=0}^{N-1} b_{ij}(t)\, x_i \quad 0 \le j \le M-1 \qquad (4-3)$$

In this equation μ_j is the output of node j and x_i is element i of the input pattern which can be 0 or 1.

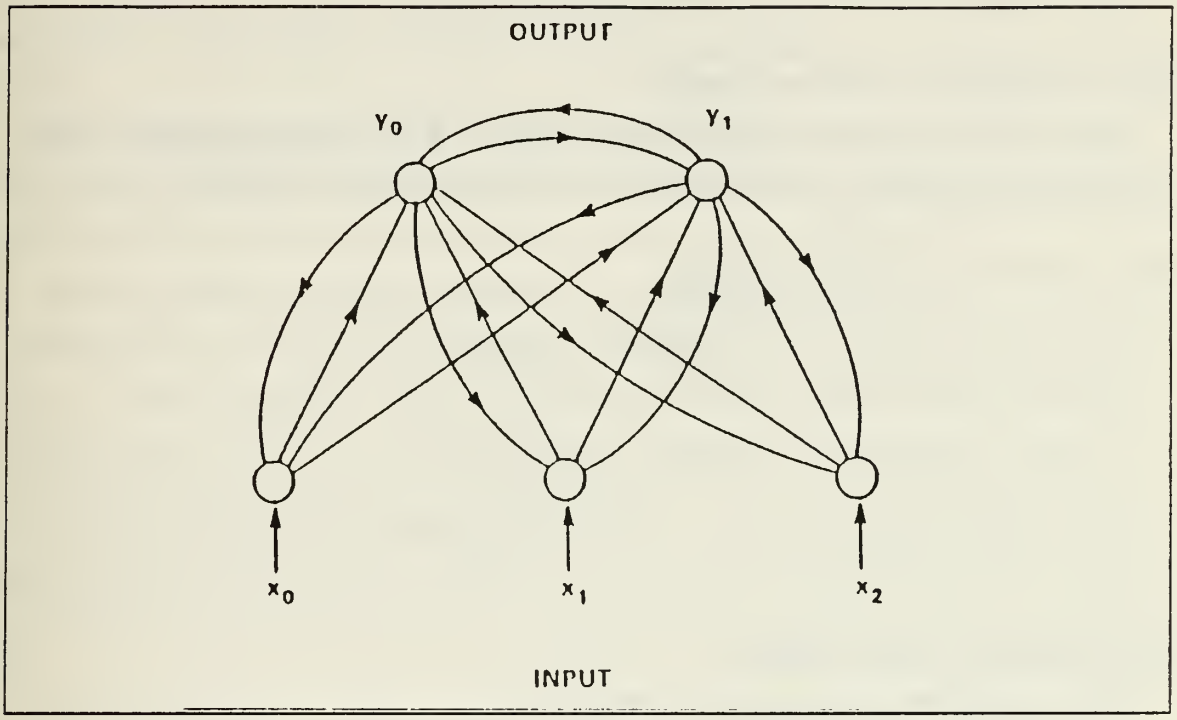


Figure 35. The major components of the Carpenter / Grossberg classification net [Ref. 2]

Step 4. Select Best Matching Exemplar

$$\mu_j^* = \max_j \{\mu_j\} \tag{4-4}$$

This is performed using extensive lateral inhibition as in the maxnet.

Step 5. Vigilance Test

$$\|X\| = \sum_{i=0}^{N-1} x_i \tag{4-5}$$

$$\|T^* X\| = \sum_{i=0}^{N-1} t_{ij^*} x_i \tag{4-6}$$

$$is \ \frac{\|T^* X\|}{\|X\|} > \rho ? \qquad (4-7)$$

If YES then GO TO Step 7, otherwise GO TO Step 6

Step 6. Disable Best Matching Exemplar

The output of the best matching node selected in Step 4 is temporarily set to zero and no longer takes part in the maximization of Step 4. Then go to Step 3.

Step 7. Adapt Best Matching Exemplar

$$t_{ij}(t+1) = t_{ij}(t)\, x_i \qquad (4-8)$$

$$b_{ij}(t+1) = \frac{t_{ij}(t)\, x_i}{0.5 + \sum_{i=0}^{N-1} t_{ij}(t)\, x_i} \qquad (4-9)$$

Step 8. Repeat by Going to Step 2

(First enable any nodes disabled in Step 6)

After initialization of the net and presentation of an unknown input pattern, matching scores are computed using feed-forward connections. The node corresponding to the exemplar with the highest matching score is selected using lateral inhibition among the output nodes as in the maxnet (Hamming net), where each output node corresponds to a stored exemplar. This net differs from the Hamming net in that feedback connections are provided from the output nodes to the input nodes and elements of both inputs and stored exemplars take on only the values 0 and 1.

The selected exemplar, from the highest matching score, is then compared to the input by computing the ratio of the dot product of the input and the best matching exemplar (number of 1 bits in common) divided by the number of 1 bits in the input. If the ratio is greater than a threshold value (vigilance) which was set at the initialization of the net (Step 1 of the algorithm), then the input is considered to be similar to the best matching exemplar and that exemplar is updated by performing a logical AND operation between its bits and those in the input. On the other hand, if the ratio is less than the vigilance threshold, the output node with the highest matching score is temporarily

set to zero, disabled by provided mechanisms. The same input pattern is presented again to the net for another test. The cycle continues until one stored exemplar matches the input or this pattern is considered to be different from all stored exemplars and it is added as a new one. Generally, when the first test fails the input is a new exemplar. Each additional exemplar requires one output node and 2N connections to compute matching scores.

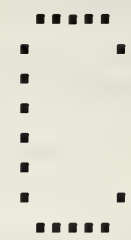
The vigilance threshold, which ranges between 0.0 and 1.0, determines how close a new pattern must be to a stored exemplar in order to be similar. A value near 1.0 means a close match is necessary and smaller values accept a poorer match.

C. SIMULATION OF THE CARPENTER / GROSSBERG NET :

Using the Fortran program provided in Appendix D, we simulated the behavior of this net. For this simulation, a vigilance threshold of 0.9 was choosen, which means that an input pattern must be very close to a stored exemplar to be considered similar. The patterns used in this simulation were of the letters "C", "E", "F" and were choosen to be of 64 element representation (matrices of 8 by 8). In all the figures provided in this discussion, we have made a black pixel to correspond to an element of value 1 and the white pixel for the value of 0. The actual input patterns used are provided in Appendix D.

Initially, the storage memory of the net was empty. To train this net, an input pattern representing the letter "C" was presented first and it was automatically stored as the net starts to learn. Now internal connection weights of the net are altered to form an internal exemplar that is identical to the letter "C" and we have the first output node of the net. In the same fashion, every learned pattern will be stored as 2N connection weights and one output node is added to the net. These 2N connections weights will form an internal exemplar for the respective pattern stored.

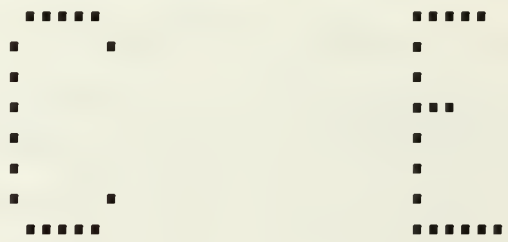
In the storage memory of the net, we have only one stored exemplar :



After, an input pattern representation of the letter "E" was applied. The response of the net to this input was :

```
BECAUSE, THE RATIO IS LESS THAN THE VIGILANCE THRESHOLD
THE INPUT PATTERN IS CONSIDERED TO BE DIFFERENT FROM
ANY EXEMPLAR PATTERN STORED. THIS INPUT PATTERN IS
THEN STORED WITH THE OTHERS AS A NEW EXEMPLAR PATTERN.
```

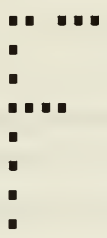
Here "E" was compared to "C" as described in Step 5 of the clustering algorithm of the net and since the ratio was less than the vigilance threshold we now have two stored exemplars.



For a new input pattern representation of the letter "F", the response of the net was the same as for "E". Here the input pattern was compared to both stored exemplars, but at all times the ratio was less than the vigilance threshold. The input pattern of "F" is then added as a new exemplar leading to three stored patterns.

At this point, we will try something different. We will present a noisy version of the letter "F" with a missing black pixel in the upper edge as shown in Appendix D. The reaction of the net was :

```
BECAUSE, THE RATIO IS GREATER THAN THE VIGILANCE
THRESHOLD, THE INPUT PATTERN IS CONSIDERED TO
MATCH A STORED PATTERN WHICH IS UPDATED BY
PERFORMING A LOGICAL 'AND' OPERATION BETWEEN
ITS BITS AND THOSE OF THE INPUT PATTERN, AND
THE NEW UPDATED PATTERN WILL LOOK LIKE:
```



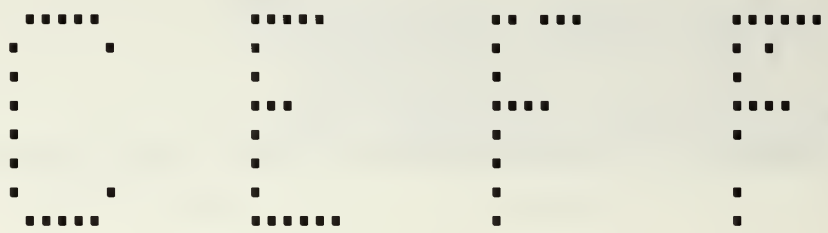
In this part of the simulation, the input pattern is found to match the stored exemplar of "F" because the ratio of the vigilance test was found greater than the vigilance value, i.e., the two patterns have many elements in common. The result was a degraded "F" due to the AND operation performed during the updating. Now in the memory of the net we still have three patterns with some changes in the pattern of the letter "F" :

Presenting an even more noisier version of the pattern of "F" given in Appendix D, the reaction of the net was :

```
BECAUSE, THE RATIO IS LESS THAN THE VIGILANCE THRESHOLD
THE INPUT PATTERN IS CONSIDERED TO BE DIFFERENT FROM
ANY EXEMPLAR PATTERN STORED. THIS INPUT PATTERN IS
THEN STORED WITH THE OTHERS AS A NEW EXEMPLAR.
```

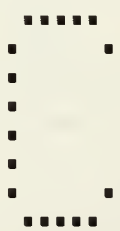
Here the input pattern was compared first to the stored pattern of the noisy "F", but the ratio was less than the vigilance value. Then it was compared to the other stored

exemplars, one in each cycle, but the ratio was still less than the vigilance value. The input pattern is then considered different from existing exemplars and it is added as a new one in the memory of the net. At this point, we have four stored patterns :



These results illustrate the inaccuracies of this net in a noisy environement. For a vigilance value of 0.9, we took a stored pattern and changed some of its elements simulating the presence of small amount of noise in the channel. Then, presenting it again as an input pattern has made the net take it as a new pattern to be stored with the correct version. Besides the noise, the value of the vigilance test can also alter the behavior of the net as we are going to show. Using the same input pattern sequence as before, we are going to simulate the clustering algorithm of the net, but this time with 0.7 as the vigilance threshold. Starting by presenting the pattern of the letter "C" as the input, the net automatically stored it in its empty memory as the net starts to learn. Now internal connections weights of the net are altered to form an internal exemplar that is identical to the letter "C".

In the storage memory of the net, we have only one stored exemplar :



Then, an input pattern representing the letter "E" was presented to net. The response of the net was:

```
BECAUSE, THE RATIO IS GREATER THAN THE VIGILANCE
THRESHOLD, THE INPUT PATTERN IS CONSIDERED
TO MATCH A STORED PATTERN WHICH IS UPDATED BY
PERFORMING A LOGICAL 'AND' OPERATION BETWEEN
ITS BITS AND THOSE OF THE INPUT PATTERN, AND
```

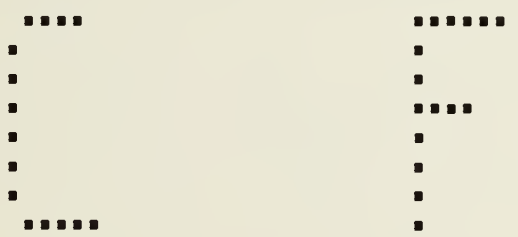
THE NEW UPDATED PATTERN WILL LOOK LIKE:

Here, comparing the input pattern "E" to the stored "C" as described in Step 5 of the clustering algorithm. The ratio was found greater than the vigilance value (0.7). The result was a degraded "C", as shown, due to the AND operation performed on its bits during the updating. In the memory of the net, the degraded pattern of "C" is stored instead of the initial pattern. Now we still have only one stored pattern.

For the pattern of the letter "F" as an input, the net responded with the following message:

```
BECAUSE, THE RATIO WAS LESS THAN THE VIGILANCE THRESHOLD
THE INPUT PATTERN IS CONSIDERED TO BE DIFFERENT FROM
ANY EXEMPLAR PATTERN STORED. THIS INPUT PATTERN IS
THEN STORED WITH THE OTHERS AS A NEW EXEMPLAR.
```

After comparing the input pattern of "F" to the stored and degraded pattern of "C" as in the vigilance test, the ratio was found to be less than the vigilance threshold which results in another stored pattern :



Now, we are going to present the noise corrupted pattern of "F". The net's response was:

```
BECAUSE, THE RATIO IS GREATER THAN THE VIGILANCE
THRESHOLD, THE INPUT PATTERN IS CONSIDERED
TO MATCH A STORED PATTERN WHICH IS UPDATED BY
```

```
PERFORMING A LOGICAL 'AND' OPERATION BETWEEN
ITS BITS AND THOSE OF THE INPUT PATTERN, AND
THE NEW UPDATED PATTERN WILL LOOK LIKE:
```

The matching pattern was of the letter "F". At this point, checking the net's memory will reveal the storage of the two patterns :

Presenting a more corrupted pattern of "F", the response of the net was as before. The input pattern was found similar to the first corrupted version of "F" (the ratio was greater than the vigilance threshold). The stored pattern of the corrupted "F" was again more disturbed after the AND operation was performed between its bits and the input pattern. The result, once again, replaced the previous version of the corrupted "F" in the memory of the net :

The results of the two simulations show clearly how the noise and the vigilance threshold can affect the performance of the Carpenter / Grossberg net. We have seen the net performing well for perfect input patterns and when adding a small amount of noise it behaves totally different. With no noise, a lower vigilance value can make the net

consider two different patterns to be similar. We have seen this in the second simulation when we have presented the pattern "E", which was mistakenly considered similar to "C" for 0.7 vigilance. On the other hand, a higher vigilance threshold can make the net consider two patterns, which are most similar, to be different. Thus, this net should not be used in a noisy channel with a high vigilance value; otherwise, the number of stored patterns will grow rapidly in time as input patterns are continuously presented until all available nodes are used up. A proportional adaptation of the vigilance threshold to the existing noise in the channel can make the net to perform perfectly during training and testing.

V. NEURAL NETWORK AS A BINARY MAXIMUM-LIKELIHOOD SEQUENCE ESTIMATOR

A. GENERALITIES :

Bandwidth-efficient data transmission over telephone and radio channels is significantly improved by the use of adaptive equalization to compensate for the time dispersion introduced by the channel.

During the last two decades, a steady research effort has produced a rich body of theory in the field of adaptive equalization and the more general field of adaptive receivers. From this work, a class of nonlinear receivers referred to as maximum-likelihood sequence estimation receivers have emerged as front-runners with respect to error rate performance. However, the high degree of computational complexity of the optimal maximum-likelihood receivers has prohibited their use in many applications. It will be shown that neural networks can be used to implement the maximum-likelihood sequence estimation and that the networks offer an attractive alternative for implementation. [Ref. 3]

Intersymbol interference caused by the bandlimiting effect of the channel is reviewed. A maximum-likelihood receiver designed to detect data symbols in the presence of intersymbol interference and additive Gaussian noise is considered and the theory behind maximum-likelihood sequence estimation is reviewed.

The maximum likelihood sequence estimation function is mapped onto a neural network structure. A neural network based receiver structure will be described which can be used for stationary or time-varying channels. The MLSE neural network will be simulated on the Mainframe and some results of its simulation will be presented.

B. MAXIMUM-LIKELIHOOD SEQUENCE ESTIMATION :

Consider a baseband synchronous data communication link used to transmit a sequence of numbers called data or information symbols, denoted by

$$\{\ldots, a_{i-1}, a_i, a_{i+1}, \ldots\}$$

The symbols are independent and can, with equal probability, be either $+1$ or -1. Let M be the number of data symbols in a transmitted sequence and assume

transmission starts at time t = 0 and ends at time t = MT. The receiver will observe the signal y(t) during the time interval starting at t = 0 and ending at $t = t_f$, where

$$t_f > (M + L)\,T$$

where L is the channel memory in units of T.

Denote the time inteval $0 \rightarrow t_f$ by I_r. By its definition, a maximum-likelihood receiver determines $\{\hat{a}_n\}$ as the best estimate sequence $\{a_n\}$ that maximizes the likelihood function $p[y(t),\ t \in I_r \mid \{a_n\}]$ given by

$$p[y(t), t \in I_r \mid \{a_n\}] \sim \exp\left\{\frac{-1}{2N_0}\int_0^{I_r}\int_0^{I_r} n(t_1 \mid \{a_n\})\, K_n^{-1}(t_1 - t_2)\, n(t_2 \mid \{a_n\})\, dt_1\, dt_2\right\} \quad (5-1)$$

where $K_n^{-1}(t)$ is the inverse of the noise autocovariance function $K_n(t)$ and

$$n(t \mid \{a_n\}) = y(t) - \sum_{k=1}^{M} a_k\, h(t - KT) \quad (5-2)$$

where h(t) is the impulse response of the matched filter used in the adaptive maximum-likelihood receivers (Figure 36). Rearranging Equation 5-2,

$$y(t) = \sum_{k=1}^{M} a_k\, h(t - KT) + n(t \mid \{a_n\}) \quad (5-3)$$

A block diagram of an adaptive maximum-likelihood receiver for the data transmission model described by Equation 5-3 is illustrated in Figure 36.

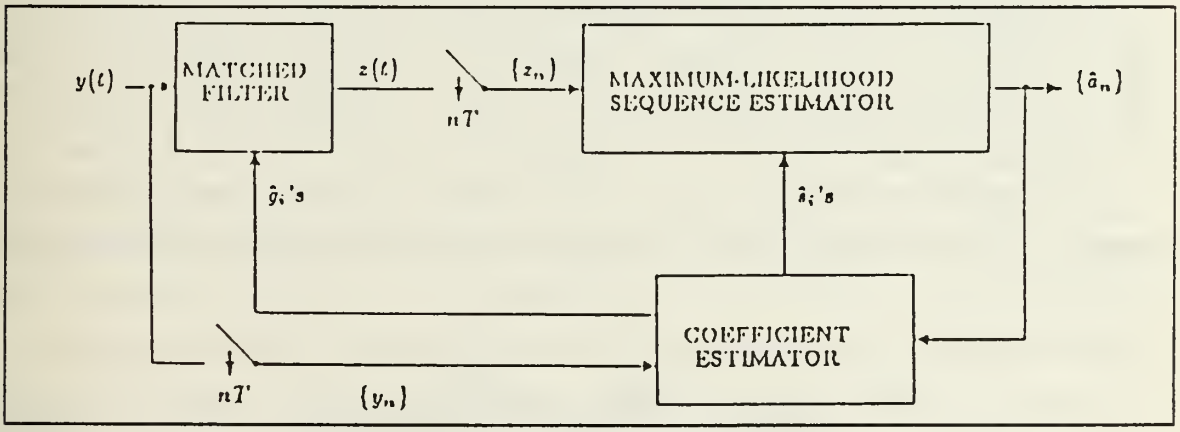


Figure 36. Adaptive Maximum-Likelihood Receiver [Ref. 3]

The impulse response of the matched filter in Figure 36, designed to improve the signal-to-noise ratio, is given by

$$g(t) = h(-t) * K_n^{-1}(t) \qquad (5-4)$$

where * denotes the convolution function. Substituting Equation 5-2 into Equation 5-1, expanding the terms in the braces and considering only terms that depend on $\{a_n\}$, yields

$$p[n(t) \mid \{a_n\}] \sim \exp\left\{\sum_{i=1}^{M} 2\, a_i\, z_i - \sum_{i=1}^{M}\sum_{k=1}^{M} a_i\, s_{i-k}\, a_k\right\} \qquad (5-5)$$

where,

$$z_n = \int_{I_r}\int_{I_r} \bar{h}(t_1 - nT)\, K^{-1}(t_1 - t_2)\, y(t_2)\, dt_1\, dt_2 \qquad (5-6)$$

$$\begin{aligned} s_l &= \int_{I_r}\int_{I_r} \bar{h}(t_1 - iT)\, K^{-1}(t_1 - t_2)\, h(t_2 - KT)\, dt_1\, dt_2 \\ &= \bar{s}_{-l} \qquad l = k - i \end{aligned} \qquad (5-7)$$

and

$$\bar{h}(t) = h(-t) \qquad (5-8)$$

The quantities z_n and s_l can be interpreted as sample values taken at the output of the matched filter, where z_n is obtained by sampling the output **z(t)** of the matched filter once every T seconds and s_l 's account for the combined response of the transmission channel and matched filter. The s_l's are symmetric and $s_l = 0$ for $|l| > L$. [Ref. 8]

Under maximum-likelihood criteria, the estimated sequence is that for which expression 5-5 is maximized. Since 5-5 is monotonically increasing function of the term in braces, given by

$$J_M(\{a_n\}) = \sum_{i=1}^{M} 2\, a_i\, z_i - \sum_{i=1}^{M}\sum_{k=1}^{M} a_i\, s_{i-k}\, a_k. \qquad (5-9)$$

maximizing Equation 5-5 is equivalent to maximizing Equation 5-9. The notation $J_M(\{a_n\})$ indicates the cost function for the sequence a_1 , a_2 , ... , a_M. Equation 5-9 will be referred to as the MLSE cost function. [Ref. 3]

The estimation procedure using direct evaluation of the MLSE cost function requires that Equation 5-9 be evaluated for all the possible sequences of length M that can be formed from data symbols +1 and −1. Thus Equation 5-9 must be evaluated 2^M times to obtain an estimate of the sequence $\{a_n\}$. To perform the estimate in real time, which is required by most communication links, the 2^M computations of Equation 5-9 must be performed in MT seconds. In most cases, direct evaluation of the MLSE cost function is too computation intensive to be of practical use. [Ref. 3]

The number of computations required can be greatly reduced by the use of the Viterbi algorithm [Refs. 8,9], which requires on the order of 2^{L-1} multiply-and-add operations during each signaling inteval T.

C. NEURAL NETWORK :

Any neural network has, as discussed before, parallel input channels, parallel output channels and a large amount of interconnections between the neural processing elements. Figure 37 illustrates the general structure of a Hopfield neural network. The processing elements (nodes), or **neurons**, are modelled as amplifiers in conjunction with feedback circuits comprised of wires, resistors and capacitors organized so as to model the most basic computational features of neurons, namely axons, dendritic arborization and synapses connecting the different neurons. [Ref. 10]

The model considered here for implementation of the MLSE neural network is that due to Hopfield and Tank [Ref. 3].

The amplifiers have sigmoid monotonic input-output relations, as shown in Figure 37. The function $v(t) = g[u(t)]$ which characterizes this nonlinear input-output relation describes the output voltage **v(t)** due to an input voltage **u(t)**. The time constants of the amplifiers are assumed negligible. However, like the input impedance caused by the cell membrane in a biological neuron, each amplifier **j** has an input resistor ρ_j leading to a reference ground and an input capacitor c_j. These components partially define the time constants of the neurons and provide for integrative analog summation of the synaptic input current from other neurons in the network.

In order to provide for both excitatory and inhibitory synaptic connections between neurons while using conventional electrical components, each amplifier is given two outputs, a normal (+) output and an inverted (−) output.

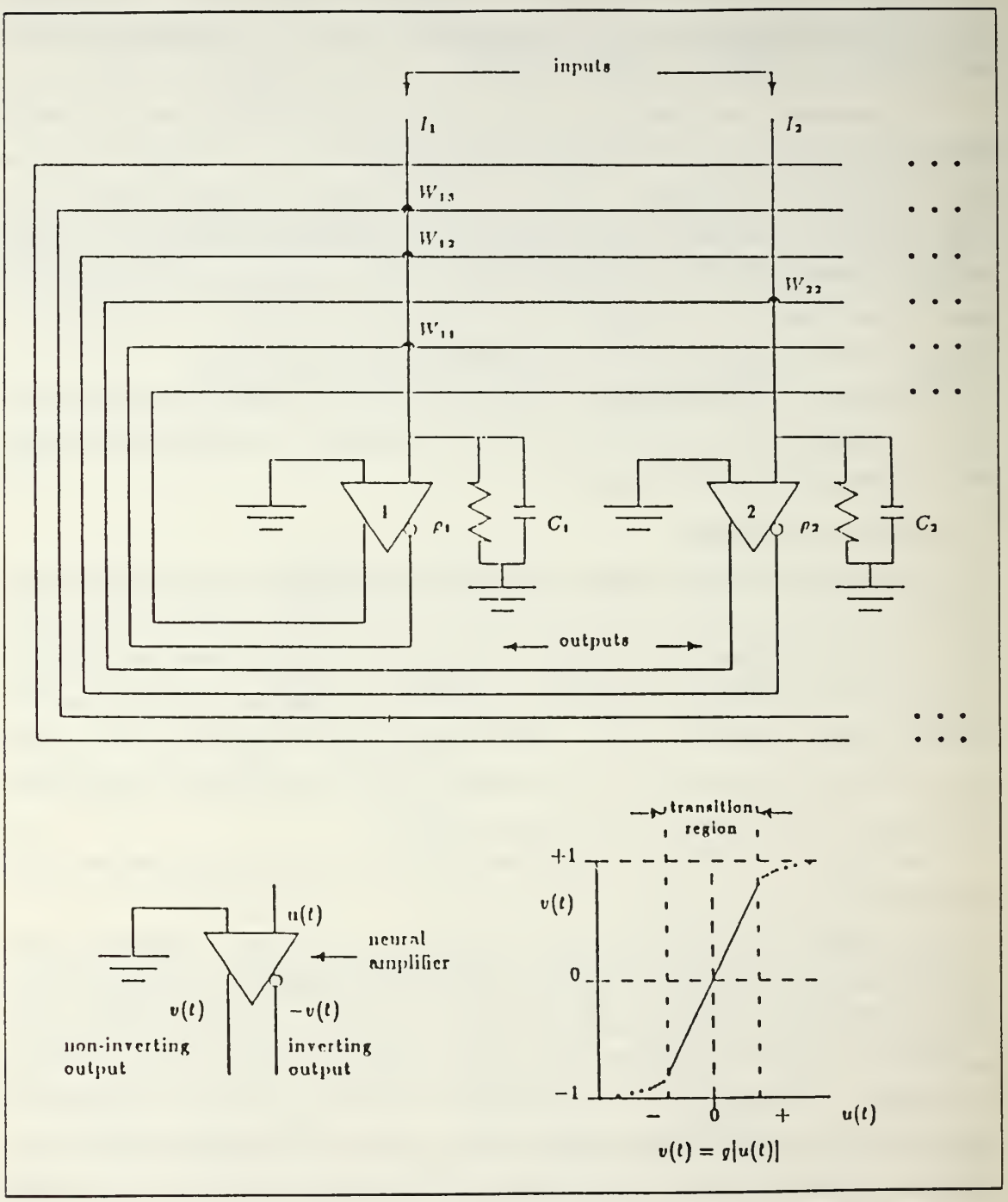


Figure 37. Hopfield Neural Network [Ref. 3]

The minimum and maximum outputs of the normal amplifier are taken as 0 and 1, while the inverted output has corresponding values of 0 and −1. A synapse between

neurons is defined by a conductance w_{ij} which connects one of the two outputs of amplifier **j** to the input of amplifier **i**. This connection is made by a resistor of value $R_{ij} = \frac{1}{|w_{ij}|}$. If the synapse is excitatory ($w_{ij} > 0$), this resistor is connected to the normal (+) output of amplifier **j**. For an inhibitory synapse ($w_{ij} < 0$), it is connected to the inverted (−) output of amplifier **j**. The matrix w_{ij} defines the connectivity among the neurons. The net input current to any neuron **i** (and hence the input voltage u_i) is the sum of the currents flowing through the set of resistors connecting its input to the outputs of the other neurons. [Ref. 9]

The set of differential equations describing the dynamics of the neural network shown in Figure 37 with M neurons are given by

$$C_j \frac{du_i(t)}{dt} = \sum_{k=1}^{M} w_{ik} v_k(t) - \frac{u_i(t)}{R_i} + I_i \quad i = 1, \ldots, M \qquad (5-10)$$

where $v_i(t) = g_i[u_i(t)]$ and R_i is the parallel combination of ρ_i and the R_{ij}'s, and C_j is the capacitance of amplifier j.

$$\frac{1}{R_i} = \frac{1}{\rho_i} + \sum_{j=1}^{M} \frac{1}{R_{ij}} \quad i = 1, \ldots, M \qquad (5-11)$$

For simplicity, we assume that $g_i[.] = g[.]$, $R_i = R$ and $C_i = C$, independent of **i**. Dividing Equation 5-10 by C and redefining $w_{ik} = \frac{w_{ik}}{C}$ and $I_i = \frac{I_i}{C}$, the equations of motion become :

$$\frac{du_i(t)}{dt} = \sum_{k=1}^{M} w_{ik} v_k(t) - \frac{u_i(t)}{\tau} + I_i \quad i = 1, \ldots, M \qquad (5-12)$$

where $\tau = RC$ is the time constant of the circuit. [Ref. 9]

In the Hopfield net operation, it was shown that the equations of motion for a network with symmetric connections ($w_{ij} = w_{ji}$) always lead to a convergence to stable states, in which the outputs of all neurons remain constant. Also, when the width of the amplifier gain curve in Figure 37 is narrow, the stable states of a neural network comprised of M neurons are the local minima of the quantity

$$E = -\sum_{i=1}^{M} v_i(t)\, I_i - \frac{1}{2}\sum_{i=1}^{M}\sum_{k=1}^{M} v_i(t)\, w_{ik}\, v_k(t) \qquad (5-13)$$

When high amplifier gain is used, the minima occur only at the corners of an M-dimensional hypercube defined by $v_i = +1$ or -1.

D. MAPPING OF MLSE ONTO A NEURAL NETWORK :

Maximizing the MLSE cost function described by Equation 5-9 is equivalent to minimizing the following expression

$$\tilde{J}_M(\{a_n\}) = -\sum_{l=1}^{M} 2\, a_i\, z_i + \sum_{i=1}^{M}\sum_{k=1}^{M} a_i\, s_{i-k}\, a_k \qquad (5-14)$$

where a_i's (of only $+1$ and -1) values which minimize Equation 5-14 are unknown. The z_i's and s_{i-k}'s are known. Comparing Equations 5-13 and 5-14 and equating variables as follows ,

$$2\, z_i = I_i\,, \quad -2\, s_{i-k} = w_{ik}\,, \quad a_i = v_i(t)$$

reveals that the two expressions are identical under these substitutions. From Equation 5-7, we recall that $s_i = s_{-i}$. Therefore,

$$w_{ik} = -2\, s_{i-k} = -2\, s_{k-i} = w_{ki}$$

which satisfies the synaptic interconnection symmetry condition. The synaptic interconnections for the neural network are determined by the coefficients which describe the combined response of the channel and matched filter. Let W denote a matrix of synaptic connections w_{ik}. Then the synaptic connection matrix W for the network is given in Figure 38.

The externally supplied input current for each neuron, I_i, is determined by observation z_i, $1 \le i \le M$. With the input voltage $u_i(t)$ initially at zero, the input sequence is applied to the network. After the network settles the estimated sequence $\{\hat{a}_n\}$ is read from the output of the neural amplifiers. A diagram of the MLSE neural network is shown in Figure 39. [Ref. 3]

$$W = \begin{bmatrix} -2s_0 & -2s_1 & \cdots & -2s_L & 0 & 0 & 0 & 0 & \cdots & 0 \\ -2s_1 & -2s_0 & -2s_1 & \cdots & -2s_L & 0 & 0 & 0 & \cdots & 0 \\ -2s_2 & -2s_1 & -2s_0 & -2s_1 & \cdots & -2s_L & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & 0 & -2s_L & \cdots & -2s_1 & -2s_0 \end{bmatrix}_{M \times M}$$

Figure 38. Matrix of synaptic connections [Ref. 3]

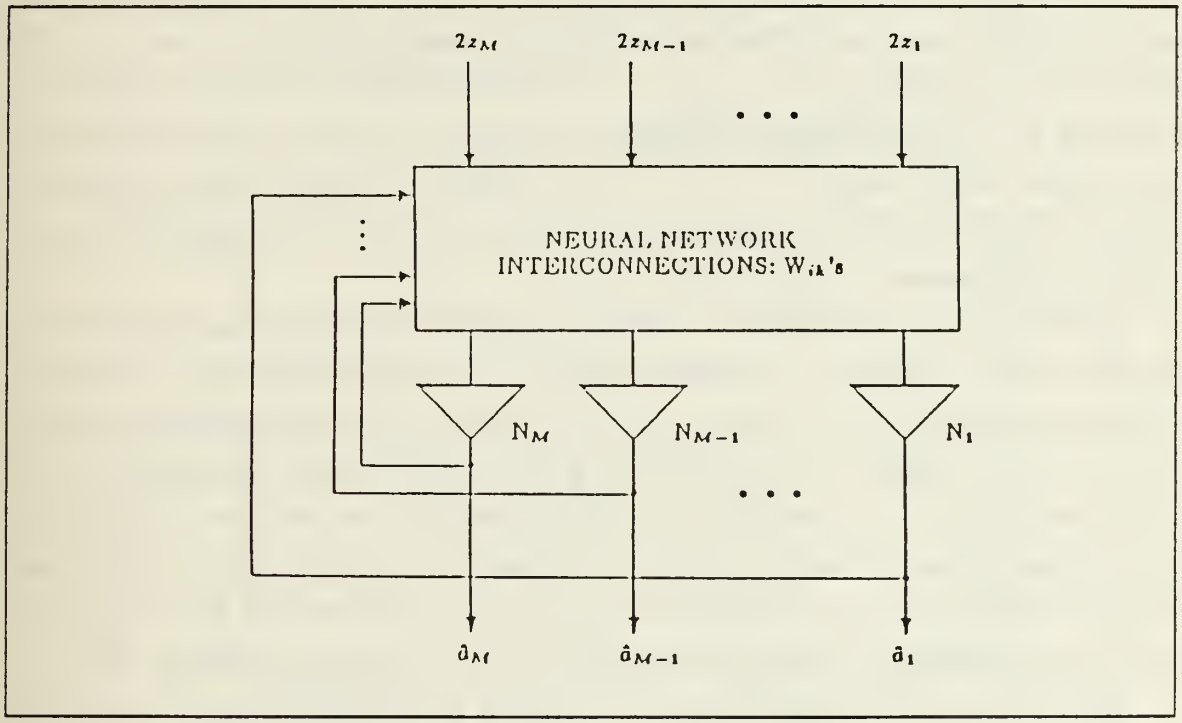


Figure 39. MLSE neural network [Ref. 3]

This development assumed that the transmission channel is stationary, which implies that the s_i's describing the combined channel and matched filter response do not change with time. Often, this is an unrealistic assumption. [Ref. 3]

The MLSE cost function given by Equation 5-14 can be written for the time-varying channel as

$$\tilde{J}_M(\{a_n\}) = -\sum_{i=1}^{M} 2\, a_i\, z_i + \sum_{i=1}^{M}\sum_{k=1}^{M} a_i\, s_{i-k}^{(i)}\, a_k \qquad (5-15)$$

From Appendix E and using Equation 5-15, the parameters for the MLSE neural network are given by [Ref. 3]

$$2\, z_i = I_i\,, \quad -2\, s'_{i-k} = -\left(s_{i-k}^{(i)} + s_{i-k}^{(k)}\right) = w_{ik}\,, \quad a_i = v_i(t) \qquad (5-16)$$

E. NEURAL NETWORK MAXIMUM-LIKELIHOOD RECEIVER :

A block diagram of the adaptive maximum-likelihood receiver incorporating the neural network for MLSE is shown in Figure 40. Registers R_1, R_2, ..., R_M form a shift register used to store the M observations. With all amplifier inputs u_i, $i = 1, \ldots, M$, initially at zero, switches SW_i, $i = 1, \ldots, M$, are simultaneously closed and the network is allowed to settle. The output of each neural amplifier is applied to the input of a decision device which outputs a $+1$ or -1 for a positive or negative input respectively. Once the network has settled, the estimated sequence is read at the output of the decision devices as shown in Figure 39. [Ref. 3]

In some cases, the length of the network, M, will be considerably less than the total number of data symbols in a transmitted sequence. For example, suppose the transmitted sequence consists of $K \times M$ data symbols. one approach is to load the first set of M observations and estimate the corresponding data symbols. After the estimate is obtained, the second set of observations would be loaded and the second set of data symbols estimated. The procedure would be performed a total of K times to obtain an estimate of the entire transmitted sequence. The primary problem with this approach is that it does not take into account the truncation of the observation sequence. [Ref. 3]

The effect of the truncation can be described by considering the role of the observations in the estimation. Let $z_i^{(k)}$, $1 \le k \le M$, denote an observation applied to the external input of neural amplifier k, where $z_i^{(k)}$ is the i^{th} observation from the received sequence.

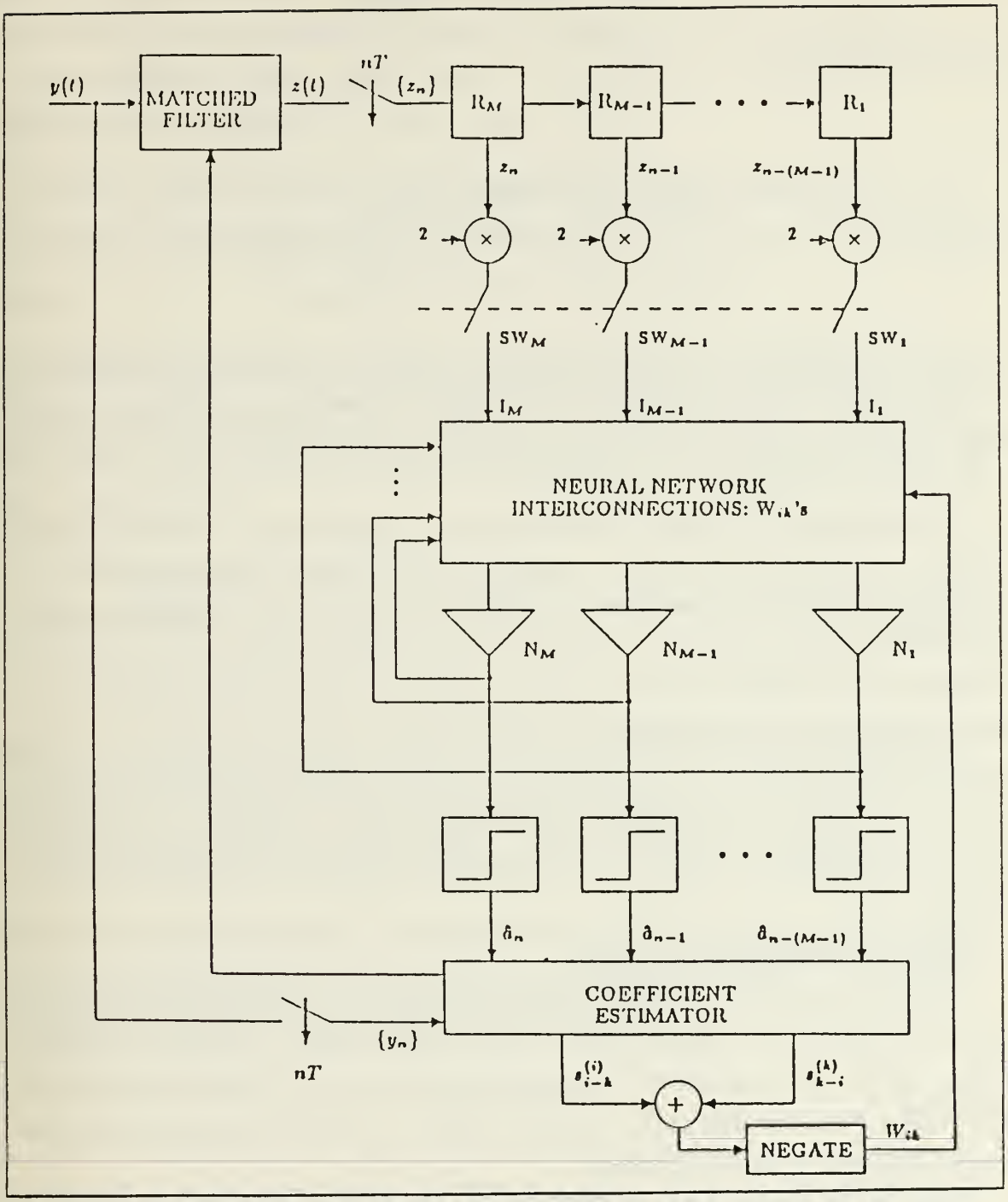


Figure 40. Neural Network Based Maximum-Likelihood Receiver [Ref. 3]

Since the channel memory is L, all information concerning the identity of data symbol a_i is contained in observations: [Ref. 3]

$$z_{i-L}^{(k-L)}, z_{i-L+1}^{(k-L+1)}, \cdots, z_{i+L}^{(k+L)}$$

for k in the interval

$$L < k < M - L + 1 \qquad (5-17)$$

All the observations containing information about the data symbol estimated by neural amplifier k are available to the network. On the other hand, for k in the intervals

$$1 \leq k \leq L \quad \& \quad M - L + 1 \leq k \leq M \qquad (5-18)$$

some of the observations containing information about the data symbol estimated by neural amplifier k are not available to the network. Therefore, one would expect more errors to occur in estimates $\hat{a}^{(k)}$ for k in the interval given by Equation 5-18 than in the interval given by Equation 5-17. [Ref. 3]

This problem can be solved by overlapping the sequences used for each estimation iteration. Assume a set of M observations have been received and the network has produced a set of M data symbol estimates. Rather than accept all M estimates as valid, only estimates from neurons L + 1 through M-1 are taken as valid. This, of course, corresponds to the estimates based on complete information about the symbols being estimated. From this set of observations, the observations in shift registers M, M-1, ..., p + 1 are saved, where $2L \leq p \leq M - 1$. A new set of p observations are shifted into the shift registers and the network performs another estimation. Essentially, this procedure amounts to shifting in p rather than M new observations after each estimation cycle. [Ref. 3]

F. SIMULATIONS AND RESULTS :

The neural network based MLSE receiver structure was implemented and simulated on the Mainframe. The program used is a self-driving program, provided as Appendix F. The network was simulated by numerically solving the set of M differential equations of Equation 5-12. The differential equations solver used in the simulation was the subroutine DGEAR of the IMSL library.

Then the M output values of DGEAR subroutine were passed through their respective neural amplifiers. The input-output function of the neural amplifiers was implemented as a hyperbolic tangent function.

$$v_i(t) = - \tanh[G\, u_i(t)] \quad i = 1, \ldots, M \qquad (5-19)$$

where G is the gain constant. Increasing G increases the slope of the input-output curve in the transition region and reduces the width of the region (see Figure 37) [Ref. 3].

The transmission channel impulse response is modeled by a finite response square cosine function given by [Ref. 10]

$$h(t,\alpha) = \begin{cases} F(\alpha)\cos^2\dfrac{\pi t}{T_0} & |t| \le \dfrac{T_0}{2} \\ 0, & |t| > \dfrac{T_0}{2} \end{cases} \qquad (5-20)$$

where the multiplicativeterm $F(\alpha)$ is included to model the time-varying channel. For the stationary channel simulations, $F(\alpha)$ is taken as 1. Channel interference also includes additive White Gaussian noise **n(t)**. The combined response of the channel and matched filter is then [Ref. 10]

$$s(t,\alpha) = \begin{cases} \dfrac{F^2(\alpha)}{2N_0}\left\{(T_0 - |t|)\left[1 + \dfrac{1}{2}\cos\left(\dfrac{2\pi t}{T_0}\right)\right] + \dfrac{3T_0}{4}\sin(2\pi|t|T_0)\right\} & |t| \le T_0 \\ 0 & |t| > T_0 \end{cases} \qquad (5-21)$$

where N_0 denotes the single-sided spectral density of the additive White Gaussian noise **n(t)**, and T_0 is the time duration of the intersymbol interference. Equations 5-20 and 5-21 are sampled at intervals of T seconds, where T is the bit duration, to generate the L + 1 discrete time channel coefficients ($h_i(\alpha)$'s) and 2L + 1 discrete time coefficients describing the combined response of the channel and matched filter ($s_i(\alpha)$'s). VLSI implementation using sequential processing techniques have been reported for data rates up to 2400 bits/second [Ref. 3]. The channel memory L is given by

$$L = \frac{T_0}{T} \qquad (5-22)$$

Actually, L is the largest integer less than or equal to $\frac{T_0}{T}$ but in this simulation we are going to take L as $\frac{T_0}{T}$.

Using the coefficients generated by sampling Equations 5-20 and 5-21 and assuming a stationary channel ($F(\alpha) = 1$) and baseband transmission model, the received samples (y_i's) are generated by the expression

$$y_i = \sum_{k=1}^{M} a_k h_{i-k} + n_i \quad i = 1, \ldots, M \qquad (5-23)$$

where y_i replaces y(t) at t = iT, h_{i-k} replaces h(t) at t = (i - k)T and n_i is a sample of the additive White Gaussian noise **n(t)** at t = iT. For this simulation,the noise samples are generated by a Gaussian random number generator GGNML of the IMSL library. The data symbols (a_k's) which are +1 and −1, are generated with equal probability using a uniform random number generator GGUD of the IMSL library. Then, the observations (z_n's) are generated by the expression

$$z_n = \sum_{k=1}^{M} y_k h_{k-n} \quad n = 1, \ldots, M \qquad (5-24)$$

The s_k's and z_n's generated by the Equations 5-23 and 5-24 are substituted into Equation 5-16 to define the parameters of the neural network. A gain factor G of 10000 was used for the simulations because of the very small output numbers of the differential equations solver DGEAR. Each simulation started with zero initial conditions and the computations were stopped after simulation of 5τ seconds (5 time constants). The estimated sequence, the M outputs of the neural amplifiers (Figure 39), was then compared to the transmitted sequence and the total number of errors were recorded. Also, the estimated sequence was compared to an estimate obtained by direct computation of the MLSE cost function and the number of data symbols which differed between the two estimated sequences was recorded.

Table 1 lists the number of neurons (M) used in the simulation, the network time constant (τ), the channel memory L which was taken to be 2 for all the simulations, the signal-to-noise ratio (SNR) given at the output of the matched filter and is computed by

$$SNR = \frac{3\,T_0}{4\,N_0}$$

the number of data (p) shifted into the registers at each simulation step, the number of symbols transmitted (N), the error data for each simulation. The last column of Table 1 lists the number of data symbol estimates which differed between the neural network estimates and direct computation of the MLSE cost function.

No. of neurons (M)	Time constant (msec)	Channel memory (L)	SNR (dB)	p	No. of symbols transmitted (N)	Total Errors	Total differences
9	29.7	2	8	6	2500	0	0
9	11.8	2	12	6	2500	0	0
9	4.7	2	16	6	2500	0	0
9	1.9	2	20	6	2500	0	0
17	14.9	2	8	12	1500	0	0
17	5.9	2	12	12	1500	0	0
17	2.3	2	16	12	1500	0	0
17	1.0	2	20	12	1500	0	0
25	9.9	2	8	18	1000	0	0
25	3.9	2	12	18	1000	0	0
25	1.6	2	16	18	1000	0	0
25	0.6	2	20	18	1000	0	0

Table 1. SIMULATIONS RESULTS FOR MLSE NEURAL NETWORK (STATIONARY CHANNEL)

To simulate the time-varying channel, the multiplicative $F(\alpha)$ will be changed at each sampling instant. The value of $F(\alpha)$ is constrained to be in the interval

$$0.8 \leq F(\alpha) \leq 1.0$$

The random number generator GGUD of the IMSL library was used to generate uniformly distributed samples, Δ_i, as described in the computer program of Appendix F. The generated samples Δ_i are distributed in the interval

$$-0.1 \leq \Delta_i \leq 0.1$$

So at a certain sampling instant, say $\alpha = iT$, the value of $F(\alpha)$ is computed by

$$F(iT) = 0.9 + \Delta_i$$

The transmission channel impulse response samples $h_k^{(p)}$'s and the combined response of the channel and matched filter $s_k^{(p)}$'s along with z_i are the parameters describing the time-varying channel at a certain time t = iT. With the exception of this modification,

the simulator for the time-varying channel is identical to that for the stationary channel. Results of this simulation for the time-varying channel are listed in Table 2. [Ref. 3]

No. of neurons (M)	Time constant (msec)	Channel memory (L)	SNR (dB)	p	No. of symbols transmitted (N)	Total Errors	Total differences
9	29.7	2	8	6	2500	0	0
9	11.8	2	12	6	2500	0	0
9	4.7	2	16	6	2500	0	0
9	1.9	2	20	6	2500	0	0
17	14.9	2	8	12	1500	0	0
17	5.9	2	12	12	1500	0	0
17	2.3	2	16	12	1500	0	0
17	1.0	2	20	12	1500	0	0

Table 2. SIMULATION RESULTS FOR MLSE NEURAL NETWORK (TIME-VARYING CHANNEL)

The neural network presented in this study can be thought of as an alternative to the Viterbi algorithm [Ref. 11] for computation of the MLSE cost function. Unlike the Viterbi algorithm implementation, the neural network does not require a vast amount of memory for storage. From the simulation results for the two transmission channel conditions for a channel memory of 2, we can conclude that the neural network can be used to estimate a transmitted sequence of binary data symbols. Comparing the estimates of the MLSE neural network and those of the direct computation of the MLSE cost function, we can say that The MLSE neural network does indeed perform the desired estimation. The amount of data provided by the simulations is far too little to make any final conclusions concerning the performance of the MLSE neural network. However, the results are promising and indicate that the neural network may be an attractive alternative for implementation of MLSE for binary communications systems. [Ref. 3]

VI. CONCLUSION

A. SUMMARY OF RESULTS :

In this study, we have first made an introduction to the field of artificial neural networks. Then, we described the use of some neural networksin pattern recognition and classification using binary pattern elements. A computer program from an algorithmic approach for each one of these networks was constructed and used to simulate the operation of the net for different cases of input pattern.

The Hopfield network was the first net we worked on. A simulation program imlementing the operation of this net as a content addressable memory for random input patterns was made. As a supervised network, the Hopfield net is only iterating between an input pattern and the ones that the teacher has already stored in its memory, showing that this net is a non-learning one. This net was simulated by presenting noise-corrupted or perfect input patterns. The response of the Hopfield net to each one of these input patterns was provided to show the iterations taken by the net to recognize and classify even noise-corrupted input patterns. By recognition and classification, we mean the net converges to one of the M stored patterns that best matches the input pattern, as long as the original pattern was stored in the net's memory prior to its use, otherwise a "no match" will occur.

However, the number of stored patterns (M) is a limitation to the proper operation of the net as a classifier because of the convergence condition demonstrated by Hopfield, which states that the net will converge with high probability if $M < 0.15N$, where N is the number of elements or bits in each pattern. These bits are taking on $+1$ and -1 values, for the $+1$ and -1 states, respectively.

The Hamming network is a classifier that calculates the Hamming distance to the exemplar of each stored class and select that class with the minimum Hamming distance to the specified input pattern. The Hamming distance is the number of bits in the input which do not match the corresponding exemplar bits. As a supervised network, we have first stored 10 exemplar patterns ($M = 10$) in its memory prior to its simulation. Simulating the operation of this network on the computer, we have seen that it effectively converges to the correct class for each input pattern. Even presenting noise corrupted input patterns, the net correctly converges to the correct class, as long as the original pattern was stored in its memory prior to the simulation, otherwise a "no match" will

occur. By convergence of the net, we mean the output nodes of the upper subnet (see Figure 8) stop changing in time and only the output node corresponding to that exemplar class which best matches the input pattern, is a positive nonzero value. While, all the other output nodes were driven to zero by inhibition. In practice, the net will converge and find the correct class when each weight w_{ij}, connection weight from input i to node j in the lower subnet (see Figure 8), is set to $\frac{1}{M-1}$ plus a small random component. Like the Hopfield net, the elements of the patterns used in these simulations were taking on $+1$ and -1 values for the $+1$ and -1 states, respectively.

As a self-organizing (a non-supervised) neural net, the Carpenter / Grossberg net self-organizes and self-stabilizes its recognition codes in response to arbitrary sequences of binary input patterns. In its learning process, the net uses a threshold level called the **vigilance value** which determines how good the learned categories will be. If vigilance value increases due to an environmental disconfirmation, then the net automatically searches for and learns the best recognition categories. The Carpenter / Grossberg net is well known as an ART system, described in Appendix C, which forms clusters and is trained without supervision. This net can learn from input patterns and later differentiate between new and learned patterns. If the new and unknown pattern is classified as previousely learned pattern at a certain level of vigilance, it will be ignored, but if it is not, it will be added as a new learned pattern. This process is repeated as long as the net is learning. The number of learned patterns thus grows with time and depends strongly on the level of vigilance used to compare input to the already stored ones. The results of simulating this network showed clearly the importance of the vigilance threshold. The first simulation was done with a vigilance value of 0.9, which means that an input pattern must be very close to a stored exemplar to be considered similar. The result was 4 patterns learned out of 6 input patterns because the net has taken two input patterns as an already learned one. Next, we have done the same simulation but this time with a vigilance value of 0.7. The results were 2 patterns learned out of 6 input patterns presented. So higher vigilance threshold can make the net to consider two patterns which are most similar, to be different and lower threshold can make the net to consider two different patterns as similar. Thus the vigilance threshold, used in the learning process of this network, is the dominant factor in the operation of this net, which behavior depends strongly on it. A proportional adaptation of the vigilance level to the existing noise in the channel can make the net to perform perfectly during training and testing. The elements of the patterns used to simulate this net take on, contrary to the Hopfield and Hamming nets, the $+1$ and 0 values for the $+1$ and -1 states, respectively.

On the other hand, bandwidth-efficient data transmission over telephone and radio channels is significantly improved by the use of adaptive equalization to compensate for the time dispersion introduced by the channel. From the work done on adaptive receivers, a class of nonlinear receivers referred to as maximum-likelihood sequence estimation receivers have emerged as front-runners with respect to error rate performance. However, the high degree of computational complexity of the optimal maximum-likelihood receivers has prohibited their use in many applications. It was shown that neural networks can be used to implement the MLSE and that these networks offer an attractive alternative for implementation. After mapping the MLSE onto a neural network, we have done some simulations on this network for stationary and time-varying channels. The results, even though they are not based on enough data to draw definitive conclusions, showed that the neural network may be an attractive alternative for implementation of the MLSE for binary communications.

B. NEURAL NETWORK TASKS :

The field of neural networks include many different models designed to address a wide range of problems in the primary application areas of *speech, vision* and *robotics.* Most researchers focus on neural networks that perform those seven major tasks illustrated graphically in Figure 41. These tasks include : [Ref. 1]

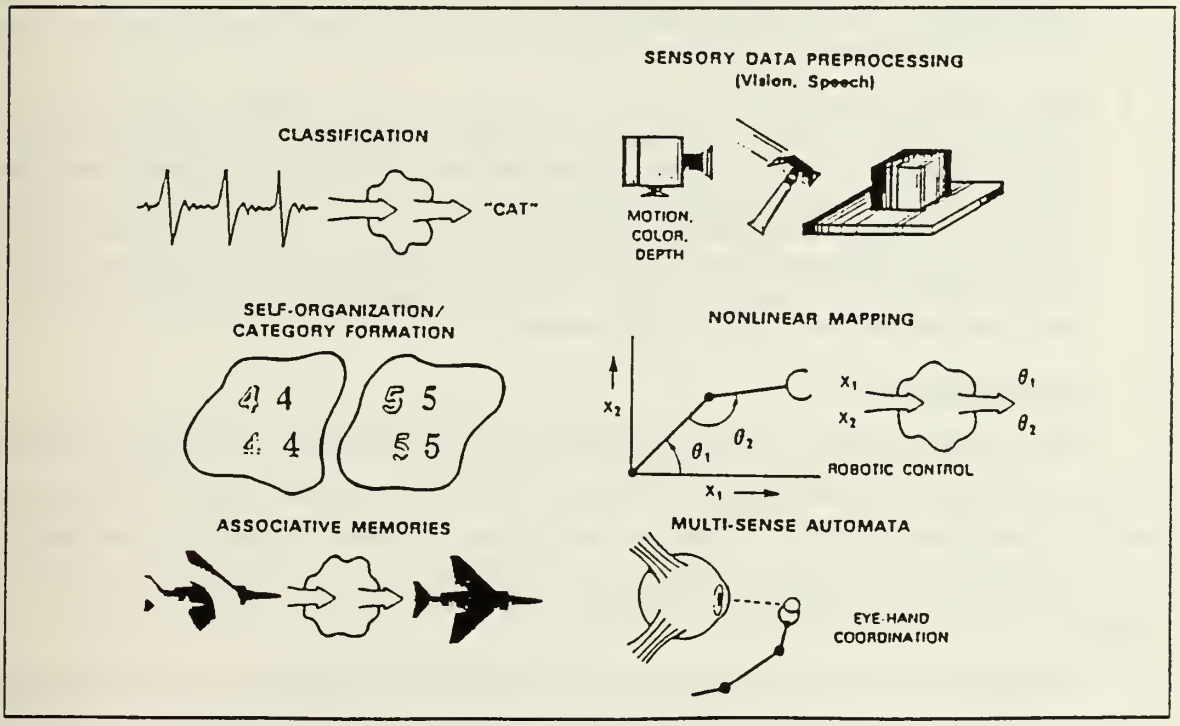


Figure 41. Seven Tasks that Neural Networks Can Perform [Ref. 1]

- **Pattern classification**: Classifiers are trained with supervision using labeled training data to partition input patterns into a pre-specified number of groups or classes. These could represent different objects for a visual image classifier. Inputs to a classifier may be binary as we have seen for the Hopfield and Hamming nets or continuous-valued.
- **Self-organization or Clustering**: Self-organizing networks, like the Carpenter / Grossberg net, partition input examples into groups or clusters using unlabeled training data. This type of clustering or vector quantization is an efficient technique for reducing information that must be processed at higher levels with little loss in performance. It also makes good use of the large amount of unlabeled training data that is typically available in speech and vision problems.
- **Associative memory (storage and access)**: An associative, or content-addressable memory provides a complete memory item from a key consisting of a partial or corrupted version of the memory. For example, it might return a complete article citation from only the author's name or a complete image of a face from only the bottom half.
- **Sensory Data Processing (vision and speech)**: An enormous amount of realtime preprocessing is performed in the peripheral sensory vision and hearing centers. Neural networks can perform this function in real time using massive parallelism.
- **Computational Problems**: Custom neural network architectures can be designed to solve specific computation problems, such as the traveling salesman problem and other constrained optimization problems, using nonlinear analog computation.
- **Nonlinear Mapping**: Many neural networks can map a vector of analog inputs into an output vector using a nonlinear mapping function which can be learned from training data. These types of mappings are useful in many areas, including robot control and nonlinear signal processing.
- **Multi-sensor Automata**: A number of complex, multi-module neural network automata have been built with visual input and a robot arm to manipulate objects in an environment. These automata demonstrate how an eye or camera can learn to scan a scene using self-supervision, how control of a multi-jointed arm and hand can then be learned using self-supervision, and then how the eye and hand can be coordinated to perform simple tasks. These automata also demonstrate how inputs from multiple sensors can be fused to provide classification performance better than could be achieved with a single sensor.

C. CONCLUSIONS :

From the study done by DARPA [Ref. 1], we can conclude that neural networks offer important new computational structures. Their real strength is derived from their ability to self-adapt and learn. If neural networks realize their full potential, they can be used for machine vision, speech recognition, signal processing, robotics and other applications.

Neural network research has matured greatly since the perceptron of 1950s, thanks to the development of advanced mathematical theories and new computer tools, and also

to a better understanding of *neurobiology*. The hardware capabilities are limiting the development of important neural network applications. It is clear that if researchers are not provided with improved simulation and implementation capabilities, the field of neural networks will once again drift off into the wilderness.

APPENDIX A. PROGRAMING THE HOPFIELD NET WHEN USED AS A CLASSIFIER :

Using Fortran as programing language, the previously described operation algorithm of the Hopfield net when used as a classifier was implemented with the mainframe, and used to run some simulations as described in the simulation paragraph of the Hopfield net.

```
C     ********************************************************************************
C     *****                      THESIS RESEARCH                                 *****
C     *****                HOPFIELD NET SIMULATION PROGRAM                       *****
C     *****                      BY M. H. KHAIDAR                                *****
C     ********************************************************************************
C
C
C     ********************************************************************************
C     *  THIS PROGRAM WAS MADE TO IMPLEMENT THE HOPFIELD NETWORK                     *
C     *  OPERATION ALGORITHM WHEN THIS NETWORK IS USED AS A                          *
C     *  CLASSIFIER. AFTER THE INPUT PATTERN IS PROCESSED AS                         *
C     *  DISSCUSSED BEFORE AND AFTER CONVERGENCE, THE OUTPUT WILL BE                 *
C     *  COMPARED TO THE M (M = 8 IN THIS IMPLEMENTATION) EXEMPLARS                  *
C     *  TO DETERMINE IF IT MATCHES AN EXEMPLAR EXACTLY. IF IT DOES,                 *
C     *  THE OUTPUT IS THAT CLASS WHOSE EXEMPLAR MATCHED THE OUTPUT                  *
C     *  PATTERN. IF IT DOES NOT THEN A "NO MATCH" RESULT OCCURS.                    *
C     *  DECLARATION OF VARIABLES:                                                   *
C     *       PATT(I,S) = THE ITH ELEMENT OF THE STH STORED EXEMPLAR                 *
C     *       T(I,J)    = THE CONNECTION WEIGHT FROM NODE I TO NODE J                *
C     *       U(J,T)    = THE OUTPUT OF NODE J AT TIME T                             *
C     *       W(I) AND V(I) = THE ITH ELEMENT IN THE MATRIX COLUNM                   *
C     *                       INPUT PATTERN W AND THE MATRIX COLUNM                  *
C     *                       OUTPUT PATTERN V                                       *
C     *       MAT(12,10), V(12,10) AND CMAT(12,10) = THE 12 BY 10                    *
C     *                       MATRIX REPRESENTATION OF AN EXEMPLAR                   *
C     *       CLASS(J)  = THE MATRIX COLUNM OF THE JTH STORED PATTERN                *
C     *       N         = THE NUMBER OF ELEMENT IN EACH EXEMPLAR                     *
C     *       M         = THE NUMBER OF STORED EXEMPLARS                             *
C     *       DIFF(120,J) = THE DIFFERENCE BETWEEN THE OUTPUT MATRIX                 *
C     *                       COLUNM AFTER CONVERGENCE AND THE JTH STORED            *
C     *                       PATTERN FOR CLASSIFICATION                             *
C     ********************************************************************************
C
      INTEGER PATT(120,8), U(10,120), T(120,120), V(12,10), CLASS(120)
      INTEGER S, I, J, K, COUNT, MAT(12,10)
      REAL W(120), VEC(120), DIFF(120,8), CMAT(12,10), DMAT(12,10)
      CHARACTER*1 TEMP(12,10)
      PRINT*,'          HOPFIELD NETWORK IMPLEMENTATION'
      PRINT*,'          ==============================='
C
```

```
C     ************************************************************************
C     *           INITIALIZE WITH UNKNOWN INPUT PATTERN                      *
C     ************************************************************************
C
      N = 120
      M = 8
      OPEN(UNIT=1, FILE='NINE1', STATUS='OLD')
      DO 5 I=1, N
           READ(1,*) W(I)
           U(1,I) = W(I)
  5   CONTINUE
      PRINT*,' '
      PRINT*,' '
      PRINT 1
  1   FORMAT('THE UNKNOWN INPUT PATTERN TO THE HOPFIELD NETWORK, '
     &'ON THE LEFT AS IM-'/
     &'POSED ON THE NETWORK AND ON THE RIGHT IN A MUCH CLEAR '
     &'REPRESENTATION'/
     &'WHERE EVERY ( * ) REPLACES A 1 AND EVERY ( . ) REPLACES A -1:')
      CALL VECMAT(W,MAT)
      PRINT*,' '
      DO 210 I=1,12
            DO 220 J=1,10
                  DMAT(I,J) = MAT(I,J)
  220       CONTINUE
  210 CONTINUE
      DO 230 I=1,12
          DO 240 J = 1,10
                IF(DMAT(I,J).EQ.1) THEN
                    TEMP(I,J) = '*'
                ELSE
                    TEMP(I,J) = '.'
                ENDIF
 240      CONTINUE
 230  CONTINUE
      DO 140 K = 1, 12
           WRITE(*,145) (MAT(K,J), J=1,10), (TEMP(K,J), J=1,10)
  140 CONTINUE
  145 FORMAT(4X,10I3,6X,10(A1,2X))
      DO 10 I=1, N
           READ(*,15) (PATT(I,S), S=1,M)
  10  CONTINUE
  15  FORMAT(1X,8I5)
C
C     ************************************************************************
C     *                  ASSIGN CONNECTION WEIGHTS                           *
C     ************************************************************************
C
      DO 20 J=1, N
           DO 30 I=1, N
                IF(I.EQ.J) THEN
                     T(I,J) = 0
                ELSE
                     SUM = 0
                     DO 35 S=1, M
                          SUM = SUM + (PATT(I,S)*PATT(J,S))
```

```
 35                CONTINUE
                   T(I,J) = SUM
                ENDIF
 30         CONTINUE
 20    CONTINUE
C
C      ***************************************************************************
C      *                ITERATE UNTIL CONVERGENCE                                *
C      ***************************************************************************
C
       DO 40 K=1, 9
            DO 50 J=1, N
                 SUM2 = 0
                 DO 60 I=1, N
                      SUM2 = SUM2 + (T(I,J)*U(K,I))
 60              CONTINUE
                 IF(SUM2.LT.0) THEN
                         U(K+1,J) = -1
                 ELSE
                         U(K+1,J) = 1
                 ENDIF
 50         CONTINUE
            FLAG = 0
            DO 70 I=1, N
                 IF(U(K,I).NE.U(K+1,I)) THEN
                      FLAG = 1
                 ENDIF
 70         CONTINUE
            IF(FLAG.EQ.0) THEN
                  COUNT = K
                  GOTO 400
            ENDIF
 40    CONTINUE
C
C      ***************************************************************************
C      *                        PRINT SHAPES                                     *
C      ***************************************************************************
C
400    DO 80 I=1, COUNT
            DO 90 J=1, N
                 VEC(J) = U(I,J)
 90         CONTINUE
            PRINT*,' '
            PRINT*,' '
            PRINT 260,I
260         FORMAT('AFTER, THE',I2,'TH ITERATION(S), THE OUTPUT OF THE '
     +      'HOPFIELD NETWORK LOOKS LIKE '/
     +      'THE FOLLOWING FOR THE '
     +      'UNKNOWN INPUT PATTERN PRESENTED:')
            PRINT*,' '
            PRINT*,' '
            CALL VECMAT(VEC,MAT)
            CALL CHARMAT(MAT,CMAT)
 80    CONTINUE
C
C      ***************************************************************************
```

```
C     *                        CLASSIFICATION                              *
C     ****************************************************************************
C
      PRINT*,' '
      PRINT*,' '
      PRINT*,'CLASSIFICATION OF THE UNKNOWN INPUT PATTERN:'
      PRINT*,'============================================'
      PRINT 2
   2  FORMAT('AT THIS POINT, FURTHER ITERATIONS WON''T MAKE ANY '
     &'CHANGE ON THE OUTPUT'/
     &'OF THE NETWORK AND THE PATTERN SPECIFIED BY THE OUTPUT NODES '
     &'IS THE'/
     &'NET''S OUTPUT. THE TASK OF THE NET NOW IS TO CLASSIFY THE '
     &'INPUT AS AN'/
     &'ALREADY KNOWN PATTERN OR A NO MATCH WILL OCCUR. AFTER '
     &'CLASSIFICATION, '/
     &'THE OUTPUT PATTERN OF THE HOPFIELD NET MATCHES BEST THE PATTERN '
     &'OF')
      DO 200 I=1, N
            CLASS(I) = U(COUNT,I)
 200  CONTINUE
      DO 180 S=1, M
            DO 190 I=1, N
                  DIFF(I,S) = PATT(I,S) - CLASS(I)
                  IF(DIFF(I,S).EQ.0) THEN
                              SCLASS = S
                  ELSE
                              GOTO 180
                  ENDIF
  190       CONTINUE
            GOTO 170
  180 CONTINUE
  170 IF(SCLASS.EQ.1)THEN
           PRINT*,'DIGIT ZERO.'
      ELSEIF(SCLASS.EQ.2)THEN
           PRINT*,'DIGIT ONE'
      ELSEIF(SCLASS.EQ.3)THEN
           PRINT*,'DIGIT TWO.'
      ELSEIF(SCLASS.EQ.4)THEN
           PRINT*,'DIGIT THREE.'
      ELSEIF(SCLASS.EQ.5)THEN
           PRINT*,'DIGIT FOUR.'
      ELSEIF(SCLASS.EQ.6)THEN
            PRINT*,'DIGIT SIX.'
      ELSEIF(SCLASS.EQ.7)THEN
           PRINT*,'BLOCK REPRESENTING THE POINT.'
      ELSEIF(SCLASS.EQ.8)THEN
           PRINT*,'DIGIT NINE.'
      ELSE
           PRINT*,'NO MATCH'
      ENDIF
      CLOSE (1)
      CLOSE (2)
      STOP
      END
C
```

```
C     ************************************************************************
c
      SUBROUTINE VECMAT(ARR,MAT)
      DIMENSION ARR(120), MAT(12,10)
      K = 0
      DO 100 J=1, 10
            DO 110 I=1,12
                  K = K + 1
                  MAT(I,J) = ARR(K)
  110       CONTINUE
  100 CONTINUE
      RETURN
      END
C
C     ************************************************************************
c
      SUBROUTINE CHARMAT(MAT,CMAT)
      DIMENSION MAT(12,10),CMAT(12,10)
      CHARACTER*1 TEMP(12,10)
      DO 120 I=1,12
            DO 130 J=1,10
                  CMAT(I,J) = MAT(I,J)
  130       CONTINUE
  120 CONTINUE
      DO 160 I=1,12
          DO 165 J = 1,10
                  IF(CMAT(I,J).EQ.1) THEN
                      TEMP(I,J) = '*'
                  ELSE
                      TEMP(I,J) = '.'
                  ENDIF
165       CONTINUE
160   CONTINUE
      DO 166 I = 1, 12
          WRITE(6,167)(TEMP(I,J),J=1,10)
167       FORMAT(22X,10(A1,2X))
166   CONTINUE
      RETURN
      END
C
C     ************************************************************************
C     * HERE ARE THE 8 STORED EXEMPLAR PATTERNS USED IN THIS PROGRAM *
C     * FOR CONVENIENCE, I CHOOSED TO WRITE THEM IN COLUNMS WHERE    *
C     * EACH ONE CORRESPONDS TO A STORED EXEMPLAR PATTERN. THE FIRST *
C     * COLUNM CORRESPONDS TO THE PATTERN OF A ZERO, THE SECOND OF A *
C     * ONE, THE THIRD OF A TWO, THE FOURTH OF A THREE, THE FIFTH OF *
C     * A FOUR, THE SIXTH OF A SIX, THE SEVENTH OF A POINT, THE EIGHT*
C     * AND LAST OF A NINE.                                          *
C     ************************************************************************
C$DATA
    -1   -1    1   -1   -1    1    1   -1
    -1   -1    1   -1   -1    1    1   -1
    -1   -1   -1   -1   -1    1    1   -1
    -1   -1   -1   -1   -1    1    1   -1
    -1   -1   -1   -1   -1    1    1   -1
    -1   -1    1   -1   -1    1    1   -1
```

-1	-1	1	-1	-1	1	-1	-1
-1	-1	1	-1	-1	1	-1	-1
-1	-1	1	-1	-1	1	-1	-1
-1	-1	1	-1	-1	1	-1	-1
-1	-1	1	-1	-1	1	-1	-1
-1	-1	1	-1	-1	1	-1	-1
-1	-1	1	-1	1	1	1	-1
-1	-1	1	-1	1	1	1	-1
-1	-1	-1	-1	1	1	1	-1
1	-1	-1	-1	1	1	1	-1
1	-1	-1	-1	1	1	1	-1
1	-1	1	-1	1	1	1	-1
1	-1	1	-1	1	1	-1	-1
1	-1	1	-1	-1	1	-1	-1
1	-1	1	-1	-1	1	-1	-1
-1	-1	1	-1	-1	1	-1	-1
-1	-1	1	-1	-1	1	-1	-1
-1	-1	1	-1	-1	1	-1	-1
-1	-1	1	1	1	1	1	-1
-1	-1	1	1	1	1	1	-1
1	-1	-1	-1	1	-1	1	-1
1	-1	-1	-1	1	-1	1	-1
1	-1	-1	-1	1	-1	1	-1
1	-1	1	-1	1	1	1	-1
1	-1	1	-1	1	1	-1	-1
1	-1	-1	-1	-1	-1	-1	-1
1	-1	-1	-1	-1	-1	-1	-1
1	-1	-1	-1	-1	-1	-1	-1
-1	-1	1	1	-1	1	-1	-1
-1	-1	1	1	-1	1	-1	-1
-1	1	1	1	-1	1	1	-1
1	1	1	1	-1	1	1	-1
1	1	-1	-1	-1	-1	1	-1
1	1	-1	-1	-1	-1	1	-1
1	1	-1	-1	-1	-1	1	-1
1	1	1	-1	1	1	1	-1
1	1	1	-1	1	1	-1	-1
1	1	-1	-1	-1	-1	-1	-1
1	1	-1	-1	-1	-1	-1	-1
1	1	-1	-1	-1	-1	-1	-1
1	1	1	1	-1	1	-1	-1
-1	1	1	1	-1	1	-1	-1
-1	1	1	1	-1	1	1	1
1	1	1	1	-1	1	1	1
1	1	-1	-1	-1	-1	1	1
-1	1	-1	-1	-1	-1	1	1
-1	1	-1	-1	-1	-1	1	1
-1	1	1	1	1	1	1	1
-1	1	1	1	1	1	-1	1
-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	1	-1	-1
1	1	-1	-1	-1	1	-1	-1
1	1	1	1	-1	1	-1	1
-1	1	1	1	-1	1	-1	1
-1	1	1	1	-1	1	-1	1
1	1	1	1	-1	1	-1	1

```
 1    1   -1   -1   -1   -1   -1    1
-1    1   -1   -1   -1   -1   -1    1
-1    1   -1   -1   -1   -1   -1    1
-1    1    1    1    1    1   -1    1
-1    1    1    1    1    1   -1    1
-1    1   -1   -1   -1    1   -1   -1
-1    1   -1   -1   -1    1   -1   -1
 1    1   -1   -1   -1    1   -1   -1
 1    1    1    1   -1    1   -1    1
-1    1    1    1   -1    1   -1    1
-1    1    1    1   -1   -1   -1    1
 1    1    1    1   -1   -1   -1    1
 1    1    1   -1   -1   -1   -1   -1
 1    1    1   -1   -1   -1   -1   -1
 1    1    1   -1   -1   -1   -1   -1
 1    1    1    1    1   -1   -1    1
 1    1    1    1    1   -1   -1    1
 1    1   -1   -1   -1   -1   -1   -1
 1    1   -1   -1   -1   -1   -1   -1
 1    1   -1   -1   -1   -1   -1   -1
 1    1    1    1   -1   -1   -1    1
-1    1    1    1   -1   -1   -1    1
-1   -1    1    1    1   -1   -1    1
-1   -1    1    1    1   -1   -1    1
 1   -1    1    1    1   -1   -1   -1
 1   -1    1    1    1   -1   -1   -1
 1   -1    1    1    1   -1   -1   -1
 1   -1    1    1    1   -1   -1    1
 1   -1    1    1    1   -1   -1    1
 1   -1   -1    1    1   -1   -1   -1
 1   -1   -1    1    1   -1   -1   -1
 1   -1   -1    1    1   -1   -1   -1
-1   -1    1    1    1   -1   -1    1
-1   -1    1    1    1   -1   -1    1
-1   -1   -1   -1    1   -1   -1    1
-1   -1   -1    1    1   -1   -1    1
-1   -1   -1    1    1   -1   -1    1
 1   -1   -1    1    1   -1   -1    1
 1   -1   -1    1    1   -1   -1    1
 1   -1   -1   -1    1   -1   -1    1
 1   -1   -1   -1    1   -1   -1    1
 1   -1   -1    1    1   -1   -1    1
 1   -1   -1    1    1   -1   -1    1
-1   -1   -1    1    1   -1   -1    1
-1   -1   -1    1    1   -1   -1    1
-1   -1   -1   -1    1   -1   -1    1
-1   -1   -1   -1   -1   -1   -1    1
-1   -1   -1   -1   -1   -1   -1    1
-1   -1   -1   -1   -1   -1   -1    1
-1   -1   -1   -1   -1   -1   -1    1
-1   -1   -1   -1   -1   -1   -1    1
-1   -1   -1   -1   -1   -1   -1    1
-1   -1   -1   -1   -1   -1   -1    1
-1   -1   -1   -1   -1   -1   -1    1
-1   -1   -1   -1   -1   -1   -1    1
```

```
-1   -1   -1   -1   -1   -1   -1    1
-1   -1   -1   -1   -1   -1   -1    1
```

APPENDIX B. PROGRAMING THE HAMMING NET WHEN USED AS AN OPTIMUM CLASSIFIER :

Using Fortran as programming language, the previously described operation algorithm of the Hamming net when used as a classifier was implemented with the Mainframe, and used to run some simulations as described in the simulation paragraph of the Hamming net.

```
C     ********************************************************************************
C     *****                      THESIS RESEARCH                                *****
C     *****                HAMMING NET SIMULATION PROGRAM                       *****
C     *****                      BY M. H. KHAIDAR                               *****
C     ********************************************************************************
C
C
C     ********************************************************************************
C     * THIS PROGRAM WAS MADE TO IMPLEMENT THE ALGORITHM OPERATION OF*
C     * THE HAMMING NET, WHEN IT IS USED AS A CLASSIFIER, PROVIDED IN*
C     * THE CHAPTER ABOUT THIS NET.                                   *
C     * VARIABLE DECLARATION :                                        *
C     *      N =  NUMBER OF NODES IN EACH EXEMPLAR                    *
C     *      M =  NUMBER OF STORED EXEMPLARS                          *
C     *      PATT(I,J) = THE ITH ELEMENT OF THE JTH STORED EXEMPLAR   *
C     *      THETA = THE THRESHOLD IN EACH NODE                       *
C     *      W(I,J) = THE CONNECTION WEIGHT FROM INPUT I TO NODE J    *
C     *      X(I) = THE ITH ELEMENT OF THE INPUT PATTERN TO THE NET   *
C     *      U(J,T) = THE OUTPUT OF NODE J AT TIME T                  *
C     *      EPSILON = THE VALUE OF WEIGHTS (INHIBITORY) BETWEEN      *
C     *                DIFFERENT OUTPUT NODES                         *
C     ********************************************************************************
C
      INTEGER PATT(120,10), U(10,11), MAT(12,10), q
      INTEGER THETA, I, J, K, T, MAP(12,10)
      REAL RESLT, EPSILON, W(120,10), SUM, SUM2, X(120), ARR(120)
      REAL CMAT(12,10)
      CHARACTER*1 TEMP(12,10)
      PRINT*,'              HAMMING NETWORK IMPLEMENTATION'
      PRINT*,'              ==============================='
      EPSILON = 0.08
C
C     ********* THETA(J) = N/2 = 120/2 = 60  ****************************
C
      THETA = 60
C
C     ********************************************************************************
C     *                     ASSIGN CONNECTION WEIGHTS                                *
C     ********************************************************************************
C
      M = 10
```

```
      N = 120
      DO 10 I=1, N
                READ(*,15) (PATT(I,J), J=1, M)
   10 CONTINUE
   15 FORMAT(1X,10I5)
      DO 20 I=1, N
           DO 25 J=1, M
                W(I,J) = PATT(I,J)/2.0
   25      CONTINUE
   20 CONTINUE
C
C     ***************************************************************************
C     *          INITIALIZATION WITH UNKNOWN INPUT PATTERN                      *
C     ***************************************************************************
C
      OPEN(UNIT=1, FILE='INPUT', STATUS='OLD')
      DO 30 I=1, N
           READ(1,*) X(I)
   30 CONTINUE
      PRINT*,' '
      PRINT*,' '
      PRINT 1
    1 FORMAT('THE UNKNOWN INPUT PATTERN TO THE HAMMING NETWORK, ON THE'
     &'LEFT AS IM-'/
     &'POSED ON THE NETWORK AND ON THE RIGHT IN A MUCH CLEAR'
     &'REPRESENTATION'/
     &'WHERE EVERY ( * ) REPLACES A 1 AND EVERY ( . ) REPLACES A -1:')
      CALL VECMAT(X,CMAT)
      CALL CHARMAT(MAT,CMAT,TEMP)
      PRINT*,' '
      PRINT*,' '
      DO 140 I =1, 12
            DO 141 J=1, 10
                  MAP(I,J) = INT(CMAT(I,J))
  141       CONTINUE
  140 CONTINUE
      DO 142 I =1, 12
            WRITE(6,145)(MAP(I,J), J=1, 10), (TEMP(I,J), J=1, 10)
  142 CONTINUE
  145 FORMAT(4X,10I3,6X,10(A1,2X))
      DO 40 J=1, M
           SUM = 0
           DO 50 I=1, N
                SUM = SUM + W(I,J)*X(I)
   50      CONTINUE
           SUM1 = SUM + THETA
           IF(SUM1.GT.0) THEN
                  U(J,1) = SUM1
           ELSE
                  U(J,1) = 0
           ENDIF
   40 CONTINUE
C
C     ***************************************************************************
C     *                  ITERATE UNTIL CONVERGENCE                              *
```

```
C     ***********************************************************************
C
      DO 60 T=1, 10
           DO 70 J=1, M
                SUM2 = 0
                DO 80 K=1, M
                     IF(K.NE.J) THEN
                          SUM2 = SUM2 + U(K,T)
                     ENDIF
  80            CONTINUE
                RESLT = U(J,T) - SUM2*EPSILON
                IF(RESLT.GT.0) THEN
                      U(J,T+1) = RESLT
                ELSE
                      U(J,T+1) = 0
                ENDIF
  70       CONTINUE
  60  CONTINUE
C
C     ***********************************************************************
C     *              THE OUTPUT OF THE HAMMING NETWORK                      *
C     ***********************************************************************
C
      PRINT*,' '
      PRINT*,' '
      PRINT 2
   2  FORMAT('THE OUTPUT OF THE HAMMING NETWORK, WHERE EACH COLUNM '
     &'REPRESENTS THE '/
     &'OUTPUT NODE VALUES FOR THE CORRESPONDING CLASSES AT A CERTAIN '
     &'NUMBER '/
     &'OF ITERATIONS:')
      PRINT*,' '
      PRINT*,' '
C
C     ***********************************************************************
C     *  ONLY TEN ITERATIONS ARE SUFFICIENT TO THE NET TO CONVERGE TO*
C     *            THE RIGHT ANSWER FOR OUR SIMULATIONS                     *
C     ***********************************************************************
C
      PRINT*,'NUMB. OF ITERATIONS= 1','    2','    3','    4','    5',
     +'    6','    7','    8','    9','   10'
      PRINT*,' '
      DO 90 J=1, M
           Q = J - 1
           WRITE(*,95) Q,(U(J,T), T=1,10)
  90  CONTINUE
  95  FORMAT(1X,'FOR CLASS',I2,':',4X,10I5)
      PRINT*,' '
      PRINT*,' '
      PRINT*,'CLASSIFICATION OF THE UNKNOWN INPUT PATTERN:'
      PRINT*,'============================================'
      PRINT 3
   3  FORMAT('THEN, THE DISTURBED UNKNOWN INPUT TO THE HAMMING NETWORK '/
     &'AFTER CONVERGENCE CORRESPONDS TO THE PATTERN STORED OF THE')
      IF(U(1,10).GT.0)THEN
            PRINT*,'CLASS ZERO.'
```

```
      ELSEIF(U(2,10).GT.0)THEN
            PRINT*,'CLASS ONE.'
      ELSEIF(U(3,10).GT.0)THEN
            PRINT*,'CLASS TWO.'
      ELSEIF(U(4,10).GT.0)THEN
            PRINT*,'CLASS THREE.'
      ELSEIF(U(5,10).GT.0)THEN
            PRINT*,'CLASS FOUR.'
      ELSEIF(U(6,10).GT.0)THEN
            PRINT*,'CLASS FIVE.'
      ELSEIF(U(7,10).GT.0)THEN
            PRINT*,'CLASS SIX.'
      ELSEIF(U(8,10).GT.0)THEN
            PRINT*,'CLASS SEVEN.'
      ELSEIF(U(9,10).GT.0)THEN
            PRINT*,'CLASS EIGHT.'
      ELSEIF(U(10,10).GT.0)THEN
            PRINT*,'CLASS NINE.'
      ENDIF
      STOP
      END
C
C     ***********************************************************************
C
      SUBROUTINE VECMAT(ARR,CMAT)
      DIMENSION ARR(120), CMAT(12,10)
      K = 0
      DO 100 J=1, 10
            DO 110 I=1,12
                  K = K + 1
                  CMAT(I,J) = ARR(K)
  110       CONTINUE
  100 CONTINUE
      RETURN
      END
C
C     ***********************************************************************
C
      SUBROUTINE CHARMAT(MAT,CMAT,TEMP)
      DIMENSION MAT(12,10), CMAT(12,10)
      CHARACTER*1 TEMP(12,10)
      DO 150 J=1, 10
            DO 160 I=1, 12
                  MAT(I,J) = CMAT(I,J)
 160        CONTINUE
 150  CONTINUE
      DO 170 I =1, 12
            DO 180 J=1, 10
                  IF(MAT(I,J).EQ.1) THEN
                        TEMP(I,J) = '*'
                  ELSE
                        TEMP(I,J) = '.'
                  ENDIF
 180        CONTINUE
 170  CONTINUE
      RETURN
```

```
      END
C
C     ***********************************************************************
C     * HERE ARE THE 10 STORED EXEMPLARS USED IN THE IMPLEMENTATION  *
C     * OF HAMMING NET. FOR CONVENIENCE, I CHOOSED TO WRITE THEM IN  *
C     * A MATRIX OF 10 COLUNMS AND 120 ROWS, WHERE EVERY COLUNM      *
C     * CORRESPONDS TO AN EXEMPLAR OF 120 ELEMENTS REPRESENTING THE  *
C     * 12 BY 10 REPRESENTATION OF THE PATTERN AS SHOWN IN THE STUDY *
C     * OF THE HAMMING NET. THE FIRST VECTOR COLUNM CORRESPONDS TO   *
C     * THE PATTERN OF DIGIT 0, THE SECOND OF 1, THE THIRD OF 2, THE *
C     * FOURTH OF 3, THE FIFTH OF 4, THE SIXTH OF 5, THE SEVENTH OF  *
C     * 6, THE EIGHT OF 7, THE NINTH OF 8 AND THE LAST OF 9.         *
C     ***********************************************************************
C$DATA
    -1   -1    1   -1   -1   -1    1   -1   -1   -1
    -1   -1    1   -1   -1   -1    1   -1   -1   -1
    -1   -1   -1   -1   -1   -1    1   -1   -1   -1
    -1   -1   -1   -1   -1   -1    1   -1   -1   -1
    -1   -1   -1   -1   -1   -1    1   -1   -1   -1
    -1   -1    1   -1   -1   -1    1   -1   -1   -1
    -1   -1    1   -1   -1   -1    1   -1   -1   -1
    -1   -1    1   -1   -1   -1    1   -1   -1   -1
    -1   -1    1   -1   -1   -1    1   -1   -1   -1
    -1   -1    1   -1   -1   -1    1   -1   -1   -1
    -1   -1    1   -1   -1   -1    1   -1   -1   -1
    -1   -1    1   -1   -1   -1    1   -1   -1   -1
    -1   -1    1   -1    1    1    1   -1   -1   -1
    -1   -1    1   -1    1    1    1   -1   -1   -1
    -1   -1   -1   -1    1    1    1   -1   -1   -1
     1   -1   -1   -1    1    1    1   -1   -1   -1
     1   -1   -1   -1    1    1    1   -1   -1   -1
     1   -1    1   -1    1    1    1   -1   -1   -1
     1   -1    1   -1    1    1    1   -1   -1   -1
     1   -1    1   -1   -1   -1    1   -1   -1   -1
     1   -1    1   -1   -1   -1    1   -1   -1   -1
    -1   -1    1   -1   -1   -1    1   -1   -1   -1
    -1   -1    1   -1   -1    1    1   -1   -1   -1
    -1   -1    1   -1   -1    1    1   -1   -1   -1
    -1   -1    1    1    1    1    1    1   -1   -1
    -1   -1    1    1    1    1    1    1    1   -1
     1   -1   -1   -1    1    1   -1   -1    1   -1
     1   -1   -1   -1    1    1   -1   -1    1   -1
     1   -1   -1   -1    1    1   -1   -1   -1   -1
     1   -1    1   -1    1    1    1   -1   -1   -1
     1   -1    1   -1    1    1    1   -1   -1   -1
     1   -1   -1   -1   -1   -1   -1   -1   -1   -1
     1   -1   -1   -1   -1   -1   -1   -1    1   -1
     1   -1   -1   -1   -1   -1   -1   -1    1   -1
    -1   -1    1    1   -1    1    1   -1    1   -1
    -1   -1    1    1   -1    1    1   -1   -1   -1
    -1    1    1    1   -1    1    1    1    1   -1
     1    1    1    1   -1    1    1    1    1   -1
     1    1   -1   -1   -1   -1   -1   -1    1   -1
     1    1   -1   -1   -1   -1   -1   -1    1   -1
     1    1   -1   -1   -1   -1   -1   -1    1   -1
     1    1    1   -1    1    1    1   -1    1   -1
```

```
 1    1    1   -1    1    1    1   -1    1   -1
 1    1   -1   -1   -1   -1   -1   -1    1   -1
 1    1   -1   -1   -1   -1   -1   -1    1   -1
 1    1   -1   -1   -1   -1   -1   -1    1   -1
 1    1    1    1   -1    1    1   -1    1   -1
-1    1    1    1   -1    1    1   -1    1   -1
-1    1    1    1   -1    1    1    1    1    1
 1    1    1    1   -1    1    1    1    1    1
 1    1   -1   -1   -1   -1   -1   -1   -1    1
-1    1   -1   -1   -1   -1   -1   -1   -1    1
-1    1   -1   -1   -1   -1   -1   -1   -1    1
-1    1    1    1    1    1    1   -1    1    1
-1    1    1    1    1    1    1   -1    1    1
-1    1   -1   -1   -1   -1    1   -1   -1   -1
-1    1   -1   -1   -1   -1    1   -1   -1   -1
 1    1   -1   -1   -1   -1    1   -1   -1   -1
 1    1    1    1   -1    1    1   -1    1    1
-1    1    1    1   -1    1    1   -1    1    1
-1    1    1    1   -1    1    1    1    1    1
 1    1    1    1   -1    1    1    1    1    1
 1    1   -1   -1   -1   -1   -1   -1   -1    1
-1    1   -1   -1   -1   -1   -1   -1   -1    1
-1    1   -1   -1   -1   -1   -1   -1   -1    1
-1    1    1    1    1    1    1   -1    1    1
-1    1    1    1    1    1    1   -1    1    1
-1    1   -1   -1   -1   -1    1   -1   -1   -1
-1    1   -1   -1   -1   -1    1   -1   -1   -1
 1    1   -1   -1   -1   -1    1   -1   -1   -1
 1    1    1    1   -1    1    1   -1    1    1
-1    1    1    1   -1    1    1   -1    1    1
-1    1    1    1   -1    1   -1    1    1    1
 1    1    1    1   -1    1   -1    1    1    1
 1    1    1   -1   -1   -1   -1    1    1   -1
 1    1    1   -1   -1   -1   -1    1    1   -1
 1    1    1   -1   -1   -1   -1    1    1   -1
 1    1    1    1    1    1   -1    1    1    1
 1    1    1    1    1    1   -1    1    1    1
 1    1   -1   -1   -1   -1   -1    1    1   -1
 1    1   -1   -1   -1   -1   -1    1    1   -1
 1    1   -1   -1   -1   -1   -1    1    1   -1
 1    1    1    1   -1    1   -1    1    1    1
-1    1    1    1   -1    1   -1    1    1    1
-1   -1    1    1    1    1   -1    1   -1    1
-1   -1    1    1    1    1   -1    1    1    1
 1   -1    1    1    1   -1   -1    1    1   -1
 1   -1    1    1    1   -1   -1    1    1   -1
 1   -1    1    1    1   -1   -1    1   -1   -1
 1   -1    1    1    1    1   -1    1   -1    1
 1   -1    1    1    1    1   -1    1   -1    1
 1   -1   -1    1    1    1   -1    1   -1   -1
 1   -1   -1    1    1    1   -1    1    1   -1
 1   -1   -1    1    1    1   -1    1    1   -1
-1   -1    1    1    1    1   -1    1    1    1
-1   -1    1    1    1    1   -1    1   -1    1
-1   -1   -1   -1    1    1   -1   -1   -1    1
-1   -1   -1    1    1    1   -1   -1   -1    1
```

-1	-1	-1	1	1	-1	-1	-1	-1	1
1	-1	-1	1	1	-1	-1	-1	-1	1
1	-1	-1	1	1	-1	-1	-1	-1	1
1	-1	-1	-1	1	1	-1	-1	-1	1
1	-1	-1	-1	1	1	-1	-1	-1	1
1	-1	-1	1	1	1	-1	-1	-1	1
1	-1	-1	1	1	1	-1	-1	-1	1
-1	-1	-1	1	1	1	-1	-1	-1	1
-1	-1	-1	1	1	1	-1	-1	-1	1
-1	-1	-1	-1	1	1	-1	-1	-1	1
-1	-1	-1	-1	-1	-1	-1	-1	-1	1
-1	-1	-1	-1	-1	-1	-1	-1	-1	1
-1	-1	-1	-1	-1	-1	-1	-1	-1	1
-1	-1	-1	-1	-1	-1	-1	-1	-1	1
-1	-1	-1	-1	-1	-1	-1	-1	-1	1
-1	-1	-1	-1	-1	-1	-1	-1	-1	1
-1	-1	-1	-1	-1	-1	-1	-1	-1	1
-1	-1	-1	-1	-1	-1	-1	-1	-1	1
-1	-1	-1	-1	-1	-1	-1	-1	-1	1
-1	-1	-1	-1	-1	-1	-1	-1	-1	1
-1	-1	-1	-1	-1	-1	-1	-1	-1	1

APPENDIX C. ART AND OPERATION OF THE CARPENTER / GROSSBERG NET :

The following is a description of the ART net operation according to Carpenter / Grossberg [Ref. 12]. A cycle that traces the real time dynamics of ART network in response to arbitrary sequences of binary input patterns is depicted in Figure 42.

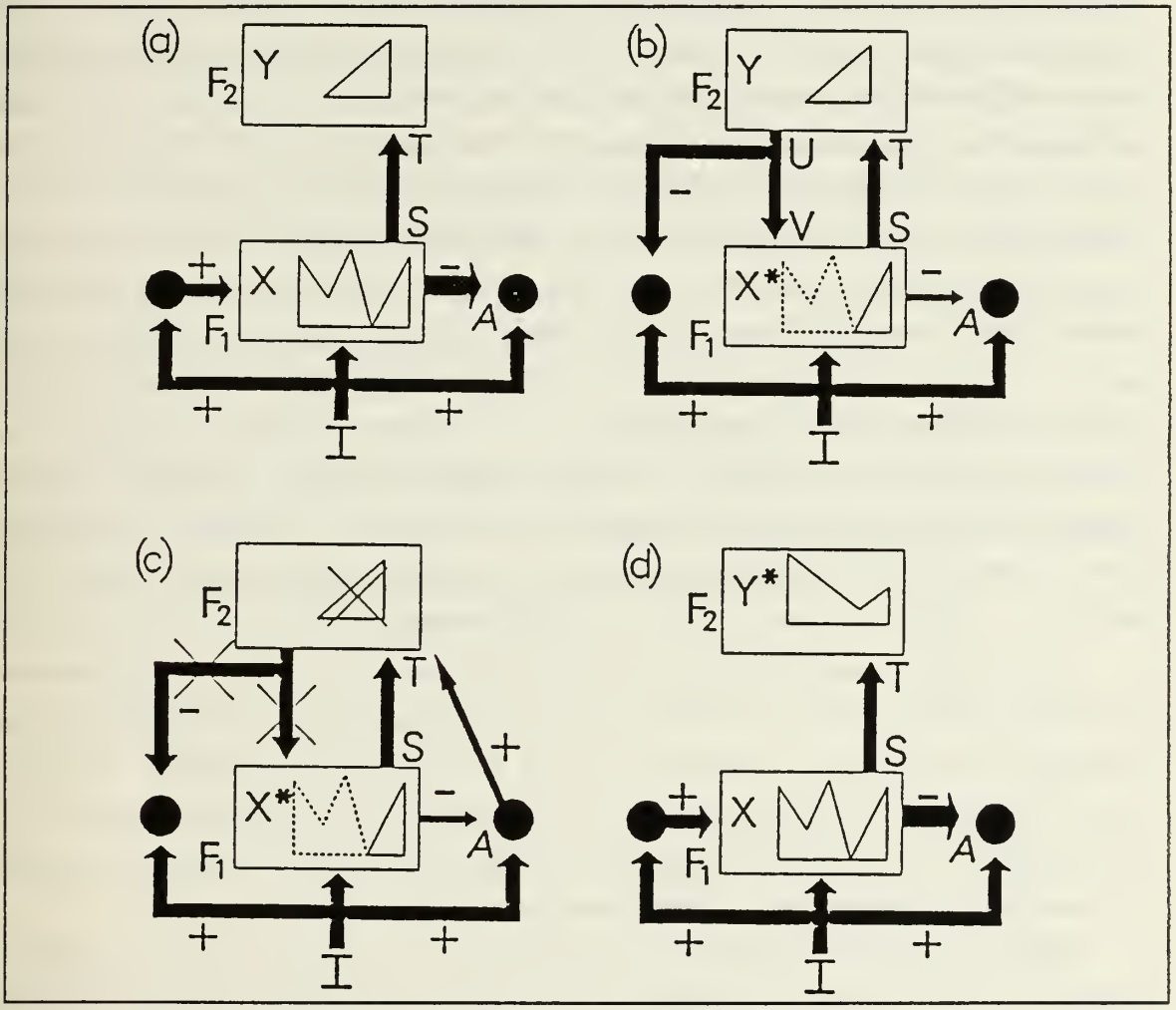


Figure 42. The ART net search for a correct F_2 code. [Ref. 12]

In Figure 42a, an unknown input pattern I is presented to the net. Pattern I is then transformed into a pattern X of activation across the nodes.In other words, the input pattern I generates a short term memory (STM) activity pattern X across a field of

feature detectors F_1 . Grossberg sees short term memory (STM) as a way of keeping patterns active after the original input pattern has vanished. A short term memory is a persistent activity pattern in a set of neurons, maintained by nonlinear feedback system.

The input pattern I also activates an orienting subsystem A, but pattern X at F_1 inhibits A before it can generate an output signal. On the other hand, the pattern X of STM activities across F_1, elicits an output pattern S of output signals from F_1. When a signal from a node in F_1 is carried along a pathway (the bottom-up adaptive filter) to F_2, the signal is multiplied or gated by the pathway's long term memory (LTM) trace. The LTM-gated signal (i.e., signal times LTM trace), not the signal alone, reaches the target node. Each target node sums up of all of its LTM-gated signals, which results in pattern S generating a pattern T of LTM-gated and summed input signals to F_2 as shown in Figure 42a. The transformation from S to T is called an adaptive filter. The input pattern T to F_2 is quickly transformed by interactions among the nodes of F_2 . The resulting pattern of activation across F_2 is a new pattern Y. This new pattern, rather than the input pattern T, is stored in STM by F_2. As soon as the bottom-up STM transformation $X \rightarrow Y$ is completed, the STM activities Y in F_2 elicit a top-down excitatory signal pattern U back to F_1 (Figure 42b). Only sufficiently large STM activities in Y elicit signals in U along the feedback pathways $F_2 \rightarrow F_1$. As before, the top-down signals U are also gated by LTM traces and the LTM-gated signals are summed at F_1 nodes. Then, the pattern U of output signals from F_2 generates a pattern V of LTM-gated and summed input signals to F_1. The transformation from U to V is thus also an adaptive filter. The pattern V is called a top-down template, or learned expectation.

Two sources of input now perturb F_1, the bottom-up input pattern I which generated the original activity pattern X and the top-down template pattern V that resulted from activating X. The amount by which activity in X is attenuated to generate X^* depends upon how much of the input pattern I is encoded within the template pattern V. In particular, F_1 acts to match V against I. Now, we will discuss how a match or mismatch of I and V at F_1 regulates the course of learning in response to the pattern I.

When a mismatch attenuates STM activity across F_1, the total size of the inhibitory signal from F_1 to A is also attenuated. If the attenuation is sufficiently great, inhibition from F_1 to A can no longer prevent the arousal source A from firing. Figure 42c shows how disinhibition of A can result in the release of an arousal burst to F_2 which equally, or nonspecifically, induces selective and enduring inhibition of active population of F_2.

In Figure 42c, inhibition of Y leads to removal of the top-down template V, and thereby terminates the mismatch between I and V. Input pattern I can thus reinstate the

original activity pattern X across F_1, which again generates the output pattern S from F_1 and the input pattern T to F_2. Due to the enduring inhibition at F_2, the input pattern T can no longer activate the original pattern Y at F_2. A new pattern Y^* is thus generated at F_2 by I (Figure 42d).

The new activity pattern Y^* reads out a new top-down template pattern V^*. If a mismatch again occurs at F_1, the orienting subsystem is again engaged, thereby leading to another arousal-mediated reset of STM at F_2. In this way, a rapid series of STM matching and reset events may occur. Such an STM matching and reset series controls the system's search of LTM by sequentially engaging the novelty-sensitive orienting subsystem. Although STM is reset sequentially in time via this mismatch mediated, self-terminating LTM search process, the mechanisms which control the LTM search are all parallel network interactions rather than serial algorithms. Such a parallel search scheme continuously adjusts itself to the system's evolving LTM codes. In general, the spatial configuration of LTM codes depends upon both the system's initial configuration and its unique learning history, and hence cannot be predicted a priori by a pre-wired search algorithm. Instead, the mismatch-mediated engagement of the orienting subsystem realizes the type of self-adjusting search.

The mismatch-mediated search of LTM ends when an STM pattern across F_2 reads-out a top-down template (V) which matches I to the degree of accuracy required by the level of attentional vigilance, or which has not yet undergone any prior learning. In this case, a new recognition category is then established and a new bottom-up code and new top-down template are learned [Ref. 6].

APPENDIX D. PROGRAMING THE CARPENTER / GROSSBERG NET

Using Fortran as programing language, the previousely described clustering algorithm of the Carpenter / Grossberg net was implemented with the Mainframe, and used to run some simulations as described in the simulation paragraph of the net.

```
C     ************************************************************************
C     *****                  THESIS RESEARCH                             *****
C     *****   CARPENTER / GROSSBERG NET SIMULATION PROGRAM               *****
C     *****                  BY  M. H. KHAIDAR                           *****
C     ************************************************************************
C
C
C     ************************************************************************
C     * THIS PROGRAM WAS MADE TO IMPLEMENT THE ALGORITHM OPERATTION         *
C     * OF THE CARPENTER / GROSSBERG NET, WHEN IT IS USED AS A CLASS-       *
C     * IFIER, PROVIDED IN THE CHAPTER FOR THIS NETWORK.                    *
C     * VARIABLE DECLARATION :                                              *
C     *     W(I,J) = THE TOP DOWN CONNECTION WEIGHT BETWEEN INPUT           *
C     *              NODE I AND OUTPUT NODE J                               *
C     *     COUNT  = THE NUMBER OF PATTERNS STORED IN THE MEMORY OF         *
C     *              THE NET AT A CERTAIN TIME T (THIS NUMBER IS VAR-       *
C     *              IANT IN TIME)                                          *
C     *     B(I,J) = THE BOTTOM UP CONNECTION WEIGHT BETWEEN INPUT          *
C     *              NODE I AND OUTPUT NODE J                               *
C     *     RO     = THE VIGILANCE THRESHOLD WHICH INDICATES HOW CLO-       *
C     *              SE AN INPUT MUST BE TO A STORED EXEMPLAR TO            *
C     *              MATCH                                                  *
C     *     PATT(I,J)= THE ITH ELEMENT OF THE JTH STORED EXEMPLAR           *
C     *     X(I)   = THE VECTOR REPRESENTATION OF THE INPUT PATTERN         *
C     *     JMAX   = THIS VARIABLE INDICATES THE CLASS THAT BEST            *
C     *              MATCHES THE INPUT PATTERN                              *
C     *     U(J,T) = THE OUTPUT OF OUTPUT NODE J AT TIME T                  *
C     ************************************************************************

      INTEGER W(64,10), ANS, COUNT, JMAX, J, I, K, PATT(64,10)
      INTEGER T, AMAT(8,8), MATRIX(8,8), TRUE, TIME
      REAL B(64,10), SUM, SUM1, SUM2, SUM3, SUM4, RO, SUM5, SUM6
      REAL EPSILON, RATIO, Y(64), RESLT, CMAT(8,8)
      REAL ARR(64), X(64), PMAT(8,8), BMAT(8,8), U(10,11)
C
C     ************************************************************************
C     *                          INITIALIZATION                              *
C     ************************************************************************
C
      PRINT*,'      CARPENTER / GROSSBERG NETWORK IMPLEMENTATION'
      PRINT*,'      ============================================'
      DO 10 I=1, 64
           DO 20 J=1, 10
                W(I,J) = 1
                B(I,J) = 0.125
```

```
 20        CONTINUE
 10    CONTINUE
       COUNT = 1
C
C      ***************************************************************************
C      *                          APPLY NEW INPUT                                *
C      ***************************************************************************
C
       OPEN(UNIT=1, FILE='LETTER', STATUS='OLD')
       DO 25 I=1, 64
            READ(1,26) PATT(I,1)
            X(I) = PATT(I,1)
  25   CONTINUE
  26   FORMAT(1X,I5)
       PRINT*,' '
       PRINT*,' '
       PRINT*,'THE FIRST INPUT PATTERN TO CARPENTER / GROSSBERG NET.:'
       CALL VECMAT(X,CMAT)
       PRINT*,' '
       PRINT*,' '
       DO 240 I=1, 8
             DO 250 J=1, 8
                   AMAT(I,J) = INT(CMAT(I,J))
 250         CONTINUE
 240   CONTINUE
       CALL CHARMAT(AMAT,PMAT)
       TRUE = 0
       JMAX = 1
       TIME = 0
       GOTO 800
  400  IF(TRUE.EQ.4) GOTO 600
       PRINT*,' '
       PRINT*,' '
       PRINT*,'PLEASE, ENTER YOUR CHOICE:'
       PRINT*,'        (1) A NEW INPUT PATTERN'
       PRINT*,' '
       PRINT*,'        (2) STOP'
       READ*,ANS
       IF(ANS.EQ.1) THEN
             TRUE = TRUE + 1
             IF(TRUE.EQ.1) THEN
                   GOTO 1
             ELSEIF(TRUE.EQ.2) THEN
                   GOTO 2
             ELSEIF(TRUE.EQ.3) THEN
                   GOTO 3
             ELSE
                   GOTO 4
             ENDIF
       ELSE
             GOTO 600
       ENDIF
 1     OPEN(UNIT=2, FILE='E', STATUS='OLD')
       DO 30 I=1, 8
            READ(2,35) (MATRIX(I,J), J=1,8)
  30   CONTINUE
```

```
 35  FORMAT(1X,8I5)
     GOTO 7
 2   OPEN(UNIT=3, FILE='F', STATUS='OLD')
     DO 31 I=1, 8
          READ(3,36) (MATRIX(I,J), J=1,8)
 31  CONTINUE
 36  FORMAT(1X,8I5)
     GOTO 7
 3   OPEN(UNIT=4, FILE='FPRIME', STATUS='OLD')
     DO 32 I=1, 8
          READ(4,37) (MATRIX(I,J), J=1,8)
 32  CONTINUE
 37  FORMAT(1X,8I5)
     GOTO 7
 4   OPEN(UNIT=5, FILE='FDPRIME', STATUS='OLD')
     DO 33 I=1, 8
          READ(5,38) (MATRIX(I,J), J=1,8)
 33  CONTINUE
 38  FORMAT(1X,8I5)
 7   DO 44 I=1, 8
          DO 45 J=1, 8
               BMAT(I,J) = REAL(MATRIX(I,J))
 45       CONTINUE
 44  CONTINUE
     CALL MATVEC(BMAT,X)
C
C    ***********************************************************************
C    *                    COMPUTE MATCHING SCORES                          *
C    ***********************************************************************
C
     DO 40 J=1, COUNT
          SUM = 0
          DO 50 I=1, 64
               SUM = SUM + B(I,J)*X(I)
 50       CONTINUE
          U(J,1) = SUM
 40  CONTINUE
C
C    ***********************************************************************
C    *              SELECT BEST MATCHING EXEMPLAR                          *
C    ***********************************************************************
C
 900 EPSILON = 0.08
     DO 60 T=1, 10
          DO 70 J=1, COUNT
               SUM1 = 0
               DO 80 K=1, COUNT
                    IF (K.NE.J) THEN
                         SUM1 = SUM1 + U(K,T)
                    ENDIF
 80            CONTINUE
               RESLT = U(J,T) - SUM1*EPSILON
               IF(RESLT.GT.0) THEN
                    U(J,T+1) = RESLT
               ELSE
                    U(J,T+1) = 0
```

```
                ENDIF
 70         CONTINUE
 60    CONTINUE
       DO 90 J=1, COUNT
            IF(U(J,9).GT.0) THEN
                 JMAX = J
            ENDIF
 90    CONTINUE
C
C      ***********************************************************************
C      *                          VIGILANCE TEST                             *
C      ***********************************************************************
C
       RO = 0.7
       SUM2 = 0
       DO 100 I=1,64
             SUM2 = SUM2 + X(I)
 100   CONTINUE
       SUM3 = 0
       DO 120 I=1, 64
             SUM3 = SUM3 + W(I,JMAX)*X(I)
 120   CONTINUE
       RATIO = SUM3 / SUM2
       IF(RATIO.GT.RO) THEN
              GOTO 200
       ELSE
              GOTO 300
       ENDIF
C
C      ***********************************************************************
C      *              DISABLE BEST MATCHING EXAMPLAR                         *
C      ***********************************************************************
C
  300 IF(TIME.NE.COUNT) THEN
            DO 46 J=1, COUNT
                 TIME = TIME + 1
                 IF(U(J,1).NE.0) THEN
                        IF(J.NE.JMAX) THEN
                             SUM5 = 0
                             DO 47 I=1, 64
                                  SUM5 = SUM5 + B(I,J)*X(I)
 47                          CONTINUE
                             U(J,1) = SUM5
                        ELSE
                             U(J,1) = 0
                        ENDIF
                 ENDIF
 46          CONTINUE
             GOTO 900
       ENDIF
       PRINT*,'BECAUSE, THE RATIO IS LESS THAN THE VIGILANCE THRESHOLD'
       PRINT*,'THE INPUT PATTERN IS CONSIDERED TO BE DIFFERENT FROM '
       PRINT*,'ANY EXAMPLAR PATTERN STORED. THIS INPUT PATTERN IS'
       PRINT*,'THEN STORED WITH THE OTHERS AS A NEW EXAMPLAR.'
       CALL VECMAT(X,CMAT)
       PRINT*,' '
```

```
      PRINT*,'THE UNKNOWN INPUT PATTERN TO CARPENTER/GROSSBERG NET.:'
      PRINT*,' '
      DO 260 I=1, 8
            DO 270 J=1, 8
                  AMAT(I,J) = INT(CMAT(I,J))
 270        CONTINUE
 260  CONTINUE
      CALL CHARMAT(AMAT,PMAT)
C
C     ***************************************************************************
C     *      ADD THE NEW INPUT PATTERN TO THE MEMORY OF THE NET      *
C     ***************************************************************************
C
      COUNT = COUNT + 1
      OPEN(UNIT=1, FILE='LETTER', STATUS='OLD')
      DO 130 I=1, 64
            PATT(I,COUNT) = X(I)
 130  CONTINUE
      REWIND 1
      DO 140 I=1, 64
            WRITE(1,145) (PATT(I,J), J=1,COUNT)
 140  CONTINUE
 145  FORMAT(1X,10I5)
      SUM6 = 0
      J = COUNT
      DO 48 I=1, 64
            SUM6 = SUM6 + W(I,J)*X(I)
 48   CONTINUE
      DO 49 I=1, 64
            B(I,J) = (W(I,J)*X(I)) / (0.5 + SUM6)
            W(I,J) = W(I,J)*X(I)
 49   CONTINUE
      TIME = 0
      GOTO 400
C
C     ***************************************************************************
C     *                ADAPT BEST MATCHING EXAMPLAR                *
C     ***************************************************************************
C
 200  PRINT*,'BECAUSE, THE RATIO IS GREATER THAN THE VIGILANCE'
      PRINT*,'THRESHOLD, THE INPUT PATTERN IS CONSIDERED '
      PRINT*,'TO MATCH A STORED PATTERN WHICH IS UPDATED BY'
      PRINT*,'PERFORMING A LOGICAL ''AND'' OPERATION BETWEEN '
      PRINT*,'ITS BITS AND THOSE OF THE INPUT PATTERN, AND'
      PRINT*,'THE NEW UPDATED PATTERN WILL LOOK LIKE:'
      DO 150 I=1, 64
            Y(I) = PATT(I,JMAX)*X(I)
 150  CONTINUE
      CALL VECMAT(Y,CMAT)
      PRINT*,' '
      DO 155 I=1, 8
            DO 157 J=1, 8
                  AMAT(I,J) = INT(CMAT(I,J))
 157        CONTINUE
 155  CONTINUE
```

```
      CALL CHARMAT(AMAT,PMAT)
C
C     ***********************************************************************
C     *  THE UPDATED PATTERN IS PUT BACK INTO THE MEMORY OF THE NET  *
C     ***********************************************************************
C
      OPEN(UNIT=1, FILE='LETTER', STATUS='OLD')
      DO 160 I=1, 64
            PATT(I,JMAX) = Y(I)
 160  CONTINUE
      REWIND 1
      DO 170 I=1, 64
            WRITE(1,175) (PATT(I,J), J=1,COUNT)
 170  CONTINUE
 175  FORMAT(1X,10I5)
 800  SUM4 = 0
      DO 180 I=1, 64
            SUM4 = SUM4 + W(I,JMAX)*X(I)
 180  CONTINUE
      DO 190 I=1, 64
            B(I,JMAX) = (W(I,JMAX)*X(I)) / (0.5 + SUM4)
            W(I,JMAX) = W(I,JMAX)*X(I)
 190  CONTINUE
      GOTO 400
 600  CLOSE (1)
      CLOSE (2)
      CLOSE (3)
      CLOSE (4)
      CLOSE (5)
      STOP
      END
C
C     ***********************************************************************
C
      SUBROUTINE VECMAT(ARR,CMAT)
      DIMENSION ARR(64), CMAT(8,8)
      K = 0
      DO 220 J=1, 8
            DO 230 I=1,8
                  K = K + 1
                  CMAT(I,J) = ARR(K)
  230       CONTINUE
  220 CONTINUE
      RETURN
      END
C
C     ***********************************************************************
C
      SUBROUTINE MATVEC(CMAT,ARR)
      DIMENSION ARR(64), CMAT(8,8)
      K = 0
      DO 310 J=1, 8
            DO 320 I=1, 8
                  K= K + 1
                  ARR(K) = CMAT(I,J)
 320        CONTINUE
```

```
 310  CONTINUE
      RETURN
      END
C
C     ***********************************************************************
C
      SUBROUTINE CHARMAT(MAT,CMAT)
      DIMENSION MAT(8,8),CMAT(8,8)
      CHARACTER*1 TEMP(8,8)
      DO 330 I=1,8
            DO 340 J=1,8
                  CMAT(I,J) = MAT(I,J)
  340       CONTINUE
  330 CONTINUE
      DO 650 I=1,8
          DO 660 J = 1,8
                  IF(CMAT(I,J).EQ.1) THEN
                      TEMP(I,J) = '*'
                  ELSE
                      TEMP(I,J) = ' '
                  ENDIF
660       CONTINUE
650   CONTINUE
      DO 370 I = 1, 8
          WRITE(6,167)(TEMP(I,J),J=1,8)
167       FORMAT(8X,10A1)
370   CONTINUE
      RETURN
      END
```

For the first input pattern to the net, we have used the pattern of the letter "C" given below. The elements of the matrice representation take on 0 and 1 values. To make the pattern clearer, we have replaced every element of 0 value by a white pixel and elements of 1 value by black pixels. A compact representation of this pattern is shown to the right below :

```
0    0    1    1    1    1    1    0          ▪▪▪▪▪
0    1    0    0    0    0    0    1        ▪      ▪
0    1    0    0    0    0    0    0        ▪
0    1    0    0    0    0    0    0        ▪
0    1    0    0    0    0    0    0        ▪
0    1    0    0    0    0    0    0        ▪
0    1    0    0    0    0    0    1        ▪      ▪
0    0    1    1    1    1    1    0         ▪▪▪▪▪
```

In a similar manner, the pattern representation of the letter "E" used in the simulation of the net is shown below. Where the left hand side pattern representation of "E" is the actual input to the net.

```
0   1   1   1   1   1   0   0        ▪▪▪▪▪
0   1   0   0   0   0   0   0        ▪
0   1   0   0   0   0   0   0        ▪
0   1   1   1   0   0   0   0        ▪▪▪
0   1   0   0   0   0   0   0        ▪
0   1   0   0   0   0   0   0        ▪
0   1   0   0   0   0   0   0        ▪
0   1   1   1   1   1   1   0        ▪▪▪▪▪▪
```

The pattern of the letter "F" is represented as :

```
0   1   1   1   1   1   1   0        ▪▪▪▪▪▪
0   1   0   0   0   0   0   0        ▪
0   1   0   0   0   0   0   0        ▪
0   1   1   1   1   0   0   0        ▪▪▪▪
0   1   0   0   0   0   0   0        ▪
0   1   0   0   0   0   0   0        ▪
0   1   0   0   0   0   0   0        ▪
0   1   0   0   0   0   0   0        ▪
```

The noise corrupted version of the pattern "F" is :

```
0   1   1   0   1   1   1   0        ▪▪ ▪▪▪
0   1   0   0   0   0   0   0        ▪
0   1   0   0   0   0   0   0        ▪
0   1   1   1   1   0   0   0        ▪▪▪▪
0   1   0   0   0   0   0   0        ▪
0   1   0   1   0   0   0   0        ▪ ▪
0   1   0   0   0   0   0   0        ▪
0   1   0   0   0   0   0   0        ▪
```

A noisier pattern of the letter "F" is :

```
0   1   1   1   1   1   1   0        ▪▪▪▪▪▪
0   1   0   1   0   0   0   0        ▪ ▪
0   1   0   0   0   0   0   0        ▪
0   1   1   1   1   0   0   0        ▪▪▪▪
0   1   0   0   0   0   0   0        ▪
0   0   0   0   0   0   0   0
0   1   0   0   0   0   0   0        ▪
0   1   0   0   0   0   0   0        ▪
```

APPENDIX E. THE PARAMETERS FOR THE MLSE NEURAL NETWORK

From [Ref. 3], the MLSE cost function for the time-varying channel can be written as

$$\tilde{J}_M(\{a_n\}) = -\sum_{i=1}^{M} 2\, a_i\, z_i + \sum_{i=1}^{M}\sum_{k=1}^{M} a_i\, s_{i-k}^{(i)}\, a_k \qquad (E-1)$$

where $s_{i-k}^{(i)}$ denotes the value of s_{i-k} at time the i^{th} observation, z_i, is sampled. The coefficients vary with the time and in general

$$s_{i-k}^{(l)} \neq s_{i-k}^{(j)} \qquad l \neq j$$

and it follows that

$$W_{ik} = s_{i-k}^{(i)} \neq s_{k-i}^{(k)} = W_{ki} \qquad i \neq k$$

thus, the symmetry condition which is sufficient for stability no longer holds.

The MLSE cost function can be reformulated such that the synaptic interconnections are symmetric. Consider the quadratic term of Equation E-1,

$$\sum_{i=1}^{M}\sum_{k=1}^{M} a_i\, s_{i-k}^{(i)}\, a_k \qquad (E-2)$$

Let α and β be two integers between 1 and M and assume for the moment that $\alpha \neq \beta$. Then two of the terms in the summation given by Equation E-2, one for i = α, k = β and the other for i = β, k = α, are

$$a_\alpha\, s_{\alpha-\beta}^{(\alpha)}\, a_\beta \quad \& \quad a_\beta\, s_{\beta-\alpha}^{(\beta)}\, a_\alpha$$

respectively. Thus, for indices α and β the summation given by Equation E-2 contains the term

$$a_\alpha\, s_{\alpha-\beta}^{(\alpha)}\, a_\beta + a_\beta\, s_{\beta-\alpha}^{(\beta)}\, a_\alpha = \frac{1}{2}\, a_\alpha\, a_\beta \left(s_{\alpha-\beta}^{(\alpha)} + s_{\beta-\alpha}^{(\beta)}\right) + \frac{1}{2}\, a_\alpha\, a_\beta \left(s_{\beta-\alpha}^{(\beta)} + s_{\alpha-\beta}^{(\alpha)}\right) \qquad (E-3)$$

The two terms on the right side of Equation E-3 are identical. Define the modified coefficient s'_{i-k} as

$$s'_{i-k} = \frac{1}{2}\left(s^{(i)}_{i-k} + s^{(k)}_{i-k}\right) \qquad (E-4)$$

clearly, the modified coefficients are symmetric independent of time in the sense that

$$s'_{i-j} = s'_{j-i}$$

When i = k, the modified coefficient becomes

$$s'_{i-i} = \frac{1}{2}\left(s^{(i)}_{i-i} + s^{(i)}_{i-i}\right) = s^{i}_{0}$$

as desired. Also, using the property given by

$$s_i = s_{-i}$$

the stationary channel case reduces to

$$s'_{i-k} = \frac{1}{2}\left(s_{i-k} + s_{k-i}\right) = s_{i-k}$$

Therefore, the MLSE cost function can be written with symmetric s'_i's in a general form suitable for either stationary or time-varying channels as

$$\tilde{J}_M(\{a_n\}) = -\sum_{i=1}^{M} 2\, a_i\, z_i + \sum_{i=1}^{M}\sum_{k=1}^{M} a_i\, s'_{i-k}\, a_k$$

where the s'_{i-k}'s are given by Equation E-4. Using the MLSE cost function for the time-varying channel, the parameters for the MLSE neural network are given by

$$2\, z_i = I_i \quad , \quad -2s'_{i-k} = -\left(s^{(i)}_{i-k} + s^{(k)}_{i-k}\right) = W_{ik} \quad , \quad a_i = v_i(t)$$

APPENDIX F. PROGRAMMING THE MLSE NEURAL NETWORK

Using Fortran as programing language, the MLSE neural network, described in Chapter V, was implemented with the Mainframe, and used to run some simulations for different network parameters and different transmission channel conditions (stationary or time-varying channel).

```
C     ************************************************************
C     *****                  THESIS RESEARCH                 *****
C     *****     SIMULATION PROGRAM OF THE MLSE NEURAL NETWORK *****
C     *****                  BY M. H. KHAIDAR                *****
C     ************************************************************
C
C     ************************************************************
C     * THIS PROGRAM WAS MADE TO IMPLEMENT AND SIMULATE THE MLSE NEU-*
C     * RAL NETWORK FOR A STATIONARY OR TIME-VARYING CHANNEL. THE PR-*
C     * OGRAM FIRST WILL ASK THE OPERATOR TO ENTER THE DATA NECESSARY*
C     * TO FULLY DESRIBE THE PROBLEM. THEN, THE PROGRAM IS GOING TO  *
C     * ASK THE OPERATOR IF THE PROGRAM IS TO BE RUN FOR A STATIONARY*
C     * CHANNEL OR A TIME-VARYING CHANNEL. AFTER THE CHOICE IS MADE  *
C     * THE COMPUTER IS GOING TO DISPLAY THE SIMULATION RESULTS.     *
C     * VARIABLE DECLARATIONS :                                      *
C     *   IR(2500) =  DATA SEQUENCE OF 2500 BITS OUTPUT OF THE GGUD  *
C     *               IMSL SUBROUTINE (= A(2600))                    *
C     *   M        =  THE NUMBER OF NEURONS IN THE NEURAL NETWORK    *
C     *               INPUT TO THE PROGRAM, THIS TIME IT IS 17.      *
C     *   MITH, MITER, INDEX, XEND, IWK(17), IER, WK(290), TOL, H=ARE*
C     *               THE DESCRIPTION PARAMETERS OF THE PROBLEM TO THE  *
C     *               DIFFERENTIAL EQUATIONS SOLVER DGEAR.           *
C     *   L        =  CHANNEL MEMORY IN UNITS OF T (L = 2 FOR THIS   *
C     *               SIMULATION), INPUT TO THE PROGRAM.             *
C     *   N        = THE NUMBER OF DATA TRANSMITTED (= NR), INPUT TO *
C     *               THE PROGRAM.                                   *
C     *   P        = THE NUMBER OF DATA BITS SHIFTED INTO THE REGIST-*
C     *               ERS AT ONCE, INPUT TO THE PROGRAM.             *
C     *   COUNT    = THE NUMBER OF DATA BITS THAT DIFFER BETWEEN THE *
C     *               TRANSMITTED DATA AND THE MLSE NEURAL NET ESTIMATED*
C     *               DATA, OUTPUT OF THE PROGRAM.                   *
C     *   NUMBER   = THE NUMBER OF DATA THAT DIFFER BETWEEN THE MLSE *
C     *               NEURAL NET ESTIMATED DATA BITS AND THE DIRECT MLSE*
C     *               COST FUNCTION CALCULATED DATA BITS, OUTPUT OF THE *
C     *               PROGRAM.                                       *
C     *   REG(17)  = DATA BITS IN THE 17 REGISTERS.                  *
C     *   G        = GAIN FACTOR OF THE NEURAL AMPLIFIERS.           *
C     *   VOUT(17) = THE DATA BITS OUTPUT OF THE NEURAL AMPLIFIERS.  *
C     *   AOUT(2500) = ALL THE VOUT(17) WILL BE COLLECTED TO FORM THE*
C     *               HOLE ESTIMATED DATA BITS CORRESPONDING TO 2500 DA-*
C     *               TA BITS TRANSMITTED.                           *
C     *   IN(2500) = THE 2500 DATA POINTS GENERATED BU GGUD SUBROUTI-*
C     *               NE TO FORM THE SAMPLES DELTA(2500) USED TO DESCRI-*
```

```
C     *            BE THE TIME-VARYING CHANNEL. (BETWEEN 1 AND 21)   *
C     *   DELTA(2500) = TIME-VARYING CHANNEL COEFFICIENTS (RANGE BET-*
C     *            -0.1 TO 0.1).                                     *
C     *   R(2500)  = THE 2500 GAUSSIAN NOISE SAMPLES USED TO IMPLEME-*
C     *            THE PRESENCE OF NOISE IN THE CHANNELS. OUTPUT OF  *
C     *            THE GGNML IMSL SUBROUTINE.                        *
C     *   U(17)    = THE 17 SOLUTIONS OF THE DIFFERENTIAL EQUATIONS  *
C     *            SOLVER DGEAR AND INPUTS TO THE NEURAL AMPLIFIERS. *
C     *   T        = TIME.                                           *
C     *   PERIOD   = BIT DURATION (INVERSE OF THE DATA RATE)         *
C     *   MUL      = A MULTIPLIER USED TO IMPLEMENT THE TIME-VARYING *
C     *            CHANNEL (MUL IS VARYING TOO).                     *
C     *   V(17)    = EQUALS MUL AT A CERTAIN REGISTER.               *
C     *   TNOT     = TIME DURATION OF THE INTERSYMBOL INTERFERENCE.  *
C     *   NNOT     = THE SINGLE SIDED SPECTRAL DENSITY OF THE ADDITI-*
C     *            VE WHITE GAUSSIAN NOISE N(T).                     *
C     *   GN(2600) = GAUSSIAN NOISE SAMPLES GENERATED BY THE GGNML   *
C     *            SUBROUTINE.                                       *
C     *   GNREG(17)= GAUSSIAN NOISE SAMLES INTO THE 17 REGISTERS OF  *
C     *            THE NEURAL NETWORK.                               *
C     *   Z(17)    = THE 17 OBSERVATIONS OF THE STATIONARY CHANNEL   *
C     *            CALCULATED AS DESCRIBED IN THE STUDY.             *
C     *   Y(17)    = THE 17 RECEIVED SAMPLES FOR A STATIONARY CHANNEL*
C     *            CALCULATED AS DESCRIBED IN THE STUDY.             *
C     *   YPRIME(2500) = EQUIVALENT TO Y(17) BUT THIS TIME WHEN      *
C     *            CALCULATING FOR THE MLSE COST FUNCTION.           *
C     *   ZPRIME(2500) = EQUIVALENT TO Z(17), FOR THE MLSE COST      *
C     *              FUNCTION.                                       *
C     *   MLSECF(2500)= THE 2500 SAMPLES GENERATED BY DIRECT CALCULA-*
C     *             TION OF THE MLSE COST FUNCTION.                  *
C     *   SNR    = SIGNAL-TO-NOISE RATIO (INPUT TO THE PROGRAM).     *
C     *   VPRIME(2500) = SAME AS V(17) BUT NOW IT'S FOR THE MLSE COST*
C     *              FUNCTION.                                       *
C     *   DR     = DATA RATE (INPUT TO THE PROGRAM, MAXIMUM 2400).   *
C     *   FCN    = SUBROUTINE DESCRIBING THE M DIFFERENTIAL EQUATIONS*
C     *   FCNJ   = EXTRA SUBROUTINE BUT NECESSARY.                   *
C     *   W(17,17) = THE SYNAPTIC CONNECTION MATRIX FOR THE NETWORK  *
C     *            ACTUALLY IT'S AN M BY M MATRIX.                   *
C     *   TAU    = TIME CONSTANT OF THE CIRCUIT.                     *
C     *   CURRENT(17) = THE 17 INPUT CURRENTS TO THE 17 NEURONS OF   *
C     *               THE NEURAL NETWORK.                            *
C     *   VIN(17) = THE 17 DATA BITS INTO THE 17 REGISTERS OF THE    *
C     *           NEURAL NETWORK.                                    *
C     ********************************************************************
C
C
      INTEGER IR(2500), X, NR, A(2600), M, METH, MITER, INDEX, ANS
      INTEGER IWK(17), IER, L, N, P, COUNT, SUP, MIN, NUMBER, RESP
      INTEGER MAX, REG(17), G, C, D, Q, VOUT(17), AOUT(2500), IN(2500)
      INTEGER DOWN, UP
      REAL R(2500), U(17), WK(290), T, TOL, H, PERIOD, MUL, V(17)
      REAL TNOT, NNOT, GN(2600), GNREG(17), SUM, FACTOR, DELTA(2500)
      REAL SUM1, Z(17), F, S, SUM2, YPRIME(2500), Y(17), SUM5, SUM6
      REAL SUM3, ZPRIME(2500), SUM4, MLSECF(2500), SNR, SUM7, SUM8
      REAL SUM9, VPRIME(2500), DR
      DOUBLE PRECISION DSEED
```

```
      EXTERNAL FCN, FCNJ
      COMMON W(17,17), TAU, CURRENT(17), VIN(17)
C
C     ***************************************************************************
      PRINT*,'         NEURAL NETWORK AS A MLSE RECEIVER'
      PRINT*,'         ================================'
      PRINT*,' '
 3000 PRINT*,'PLEASE, ENTER YOUR CHOICE, DO YOU WANT TO :'
      PRINT 1
  1   FORMAT(8X,'(1)',' ','CONTINUE WITH THE PROGRAM'/
     &8X,'(2)',' ','QUIT')
      READ*,RESP
      IF(RESP.EQ.1) THEN
           GOTO 2
      ELSE
           GOTO 3
      ENDIF
  2   PRINT*,' '
      PRINT*,' '
      PRINT*,'PLEASE, ENTER THE NUMBER OF NEURONS (M) :'
      READ*,M
      PRINT*,'PLEASE, ENTER THE DATA RATE DESIRED (HZ) :'
      READ*,DR
      PRINT*,'PLEASE, ENTER THE CHANNEL MEMORY (L) :'
      READ*,L
      PRINT*,'PLEASE, ENTER THE TIME CONSTANT (TAU) :'
      READ*,TAU
      PRINT*,'PLEASE, ENTER THE SNR (IN DB) :'
      READ*,SNR
      PRINT*,'PLEASE, ENTER THE NUMBER OF SYMBOL TRANSMITTED (N) :'
      READ*,N
      PRINT*,'PLEASE, ENTER THE NO. OF SHIFTED SYMBOLS IN THE REG. (P):'
      READ*,P
      COUNT = 0
      DOWN = 1
      UP = M
      SUP = M
      MIN = 1
      NUMBER = 0
      MAX = M
C
C     ***************************************************************************
C     * GENERATION OF THE N DATA SEQUENCE AND THE N GAUSSIAN NOISE   *
C     * SAMPLES AND PUTTING THE FIRST M DATA BITS AND M NOISE SAMPLES*
C     * INTO THE M REGISTERS OF THE NETWORK TO START THE PROCESSING. *
C     ***************************************************************************
C
      PERIOD = 1 / DR
      TNOT = L * PERIOD
      NNOT = (3 * TNOT) / (4 * (10 ** (SNR / 10.0)))
      X = 2
      NR = N
      DSEED = 123457.0D00
      CALL GGUD(DSEED,X,NR,IR)
      DO 10 I = 1, NR
           IF(IR(I).EQ.2) THEN
```

```
                A(I) = -1
            ELSE
                A(I) = +1
            ENDIF
   10 CONTINUE
      DSEED = 123457.0D00
      CALL GGNML(DSEED,NR,R)
      DO 20 I = 1, NR
           GN(I) = R(I)
   20 CONTINUE
      DO 30 I = 1, M
           REG(I) = A(I)
           GNREG(I) = GN(I)
   30 CONTINUE
      PRINT*,' '
      PRINT*,' '
      PRINT*,'YOUR DATA IS NOW ENTERED, THE NETWORK IS READY TO BE'
      PRINT*,'SIMULATED. THIS PROGRAM CAN SIMULATE THE MLSE NEURAL'
      PRINT*,'NETWORK IN TWO CONDITIONS OF TRANSMISSION CHANNEL :'
      PRINT*,' '
      PRINT 5
    5 FORMAT(8X,'(1)',' ','IN A STATIONARY CHANNEL'/
     &8X,'(2)',' ','IN A TIME-VARYING CHANNEL')
      PRINT*,' '
      PRINT*,'PLEASE, ENTER YOUR CHOICE :'
      READ*,ANS
      PRINT*,' '
      PRINT*,' '
      PRINT 12
   12 FORMAT(12X,'PROCESSING IN PROGRESS ..... PLEASE WAIT')
      PRINT*,' '
      PRINT*,' '
      IF(ANS.EQ.1) THEN
             GOTO 200
      ELSE
             GOTO 500
      ENDIF
C
C     ***********************************************************************
C     *   SIMULATION OF MLSE NEURAL NETWORK IN A STATIONARY CHANNEL  *
C     ***********************************************************************
C
  200 DO 40 I = 1, M
           SUM = 0
           DO 50 K = 1, M
                FACTOR = (I - K) * PERIOD
                SUM = SUM + REG(K) * F(FACTOR,TNOT)
   50      CONTINUE
           Y(I) = SUM + GNREG(I)
   40 CONTINUE
      DO 60 I = 1, M
           SUM1 = 0
           DO 70 K = 1, M
                FACTOR = (K - I) * PERIOD
                SUM1 = SUM1 + Y(K) * F(FACTOR,TNOT)
   70      CONTINUE
```

```
            Z(I) = SUM1
    60 CONTINUE
       DO 90 I = 1, M
             CURRENT(I) = 2 * Z(I)
             VIN(I) = REG(I)
             DO 100 K = 1, M
                   FACTOR = (I - K) * PERIOD
                   W(I,K) = -2 * S(FACTOR,TNOT,NNOT)
   100       CONTINUE
    90 CONTINUE
       GOTO 1000
C
C     ***********************************************************************
C     *  SIMULATION OF MLSE NEURAL NETWORK IN A TIME-VARYING CHANNEL *
C     ***********************************************************************
C
   500 NB = 21
       NR = N
       DSEED = 123457.0D0
       CALL GGUD(DSEED,NB,NR,IN)
       DO 310 I = 1, NR
             IF(IN(I).GE.1.AND.IN(I).LT.11) THEN
                   DELTA(I) = -IN(I) / 100.0
             ELSEIF(IN(I).GE.11.AND.IN(I).LT.21) THEN
                   DELTA(I) = ( IN(I) - 10 ) / 100.0
             ELSE
                   DELTA(I) = 0
             ENDIF
   310 CONTINUE
       K = 1
       DO 320 I = DOWN , UP
             V(K) = DELTA(I) + 0.9
             K = K + 1
   320 CONTINUE
       DOWN = DOWN + M
       UP = UP + M
       IF(UP.GE.N) UP = N
       DO 330 I = 1 , M
             SUM5 = 0
             DO 340 K = 1 , M
                   FACTOR = (I - K) * PERIOD
                   MUL = V(K)
                   SUM5 = SUM5 + REG(K) * FPRIME(FACTOR,TNOT,MUL)
   340       CONTINUE
             Y(I) = SUM5 + GNREG(I)
   330 CONTINUE
       DO 460 I = 1 , M
             SUM6 = 0
             DO 360 K = 1 , M
                   FACTOR = (K - I) * PERIOD
                   MUL = V(K)
                   SUM6 = SUM6 + Y(K) * FPRIME(FACTOR,TNOT,MUL)
   360       CONTINUE
              Z(I) = SUM6
   460 CONTINUE
       DO 370  I = 1 , M
```

```
              CURRENT(I) = 2 * Z(I)
              VIN(I) = REG(I)
              DO 380 K = 1 , M
                    FACTOR = (I - K) * PERIOD
                    MUL = V(K)
                    W(I,K) = -2 * SPRIME(FACTOR,TNOT,NNOT,MUL)
  380          CONTINUE
  370 CONTINUE
 1000 G = 10000
      T = 0.0
      DO 80 I = 1, M
           U(I) = 0.0
   80 CONTINUE
      TOL = 0.00001
      H = 0.000001
      MITER = 0
      METH = 1
      INDEX = 1
      XEND = 5 * TAU
C
C     ***************************************************************
C     * AFTER DOING SOME CALCULATIONS NOW WE ARE READY TO CALL DGEAR *
C     * THE DIFFERENTIAL EQUATIONS SOLVER TO SOLVE OUR M EQUATIONS.  *
C     ***************************************************************
C
      CALL DGEAR(M,FCN,FCNJ,T,H,U,XEND,TOL,METH,MITER,INDEX,IWK,WK,IER)
      IF(IER.GT.128) THEN
             PRINT 13,IER
   13        FORMAT(/,'   WARNING !!!!!...IER =    ',I5)
             GOTO 3
      ENDIF
C
C     ***************************************************************
C     *THE OUTPUT OF DGEAR, SOLUTIONS TO OUR M DIFFERENTIAL EQUATIONS*
C     *ARE PASSED THROUGH THE M NEURAL AMPLIFIERS TO GET THE MLSE NEU*
C     *RAL NETWORK ESTIMATES OF THE M DATA BITS THAT ARE IN THE M REG*
C     *ISTERS.                                                      *
C     ***************************************************************
C
      DO 110 I = 1, M
           VOUT(I) = -TANH(G * U(I))
  110 CONTINUE
      DO 111 I = 1 , M
            IF(REG(I).EQ.0) VOUT(I) = 0
  111 CONTINUE
C     ***************************************************************
C     *HERE THE DATA BITS OUTPUTS OF THE NEURAL AMPLIFIERS ARE COLLE-*
C     *CTED SO THAT LATER WE ARE GOING TO HAVE THE HOLE 2500 ESTIMATE*
C     *S OF THE 2500 DATA BITS TRANSMITTED.                         *
C     ***************************************************************
C
      IF(MAX.EQ.M) THEN
            K = 1
            DO 115 J = 1 , M-1
                  AOUT(K) = VOUT(J)
                  K = K + 1
```

```
  115       CONTINUE
      ELSEIF(MAX.NE.N) THEN
            DO 116 J = M-P , M-1
                  AOUT(K) = VOUT(J)
                  K = K + 1
  116       CONTINUE
      ELSEIF(MAX.EQ.N) THEN
           MPRIME = M - P
           DO 118 I = M , 1 , -1
                 IF(REG(I).EQ.0) THEN
                       M = M - 1
                 ENDIF
  118       CONTINUE
            IF(M.GT.MPRIME) THEN
                 DO 117 J = MPRIME , M
                       AOUT(K) = VOUT(J)
                       K = K + 1
  117            CONTINUE
            ENDIF
      ENDIF
C
C     ***************************************************************************
C     *HERE, THE ESTIMATES FROM NEURONS L+1 THROUGH M-1 ARE TAKEN AS *
C     *VALID. THEN A COMPARISON BETWEEN THESE ESTIMATES AND THEIR COR*
C     *RESPONDING IN THE REGISTERS I.E. THE INPUT DATA BITS IS MADE  *
C     *AND THE RESULT IS RECORDED FOR LATER USE.                     *
C     ***************************************************************************
C
      C = L + 1
      D = M - 1
      IF(MAX.EQ.M) THEN
            DO 120 I = 1, M
                  IF(VIN(I).NE.VOUT(I)) THEN
                        COUNT = COUNT + 1
                  ENDIF
  120       CONTINUE
      ELSEIF(MAX.EQ.N) THEN
            DO 165 I = M, 1, -1
                  IF(REG(I).EQ.0) THEN
                        M = M - 1
                  ENDIF
  165       CONTINUE
            DO 166 I = C, M
                  IF(VIN(I).NE.VOUT(I)) THEN
                        COUNT = COUNT + 1
                  ENDIF
  166       CONTINUE
      ELSE
            DO 130 I = C, D
                  IF(VIN(I).NE.VOUT(I)) THEN
                        COUNT = COUNT + 1
                  ENDIF
  130       CONTINUE
      ENDIF
C
C     ***************************************************************************
```

```
C     *HERE, THE DATA BITS IN THE REGISTERS FROM 1 TO P ARE NULLED   *
C     *THEN THE CONTENT OF THE SHIFT REGISTERS ARE SHIFTED TO THE RIG*
C     *HT TILL THE CONTENT OF REGISTER 1 IS NONZERO. THEN, WE ARE FEE*
C     *DING THE P EMPTY REGISTERS BY THE NEXT P DATA BITS OF THE TRAN*
C     *SMITTED SEQUENCE FOR ANOTHER PROCESSING CYCLE.                 *
C     ****************************************************************
C
      DO 140 I = 1, P
            REG(I) = 0
            GNREG(I) = 0
  140 CONTINUE
      Q = 1
  150 IF(Q.LE.M) THEN
         IF((P+Q).LE.M) THEN
            REG(Q) = REG(P+Q)
            GNREG(Q) = GNREG(P+Q)
         ELSE
            REG(Q) = 0
            GNREG(Q) = 0
         ENDIF
         Q = Q + 1
         GOTO 150
      ENDIF
  155 IF(MAX.EQ.N) THEN
      PRINT*,'THE NUMBER OF ERROR DATA BETWEEN THE TRANSMITTED'
      PRINT*,'BINARY SEQUENCE AND THE MLSE NEURAL NET OUTPUT'
      PRINT*,'DATA IS :'
      PRINT*,' '
      PRINT 270, COUNT
  270 FORMAT(2X,'COUNT =',2X,I5)
      GOTO 2000
      ENDIF
      DO 160 I = 1, M
            IF(REG(I).EQ.0) THEN
                  MAX = MAX + 1
                  IF(MAX.GT.N) THEN
                       A(MAX) = 0
                       GN(MAX) = 0
                  ENDIF
                  REG(I) = A(MAX)
                  GNREG(I) = GN(MAX)
            ENDIF
  160 CONTINUE
      IF(MAX.GT.N) THEN
            MAX = N
      ENDIF
      IF(ANS.EQ.1) THEN
             GOTO 200
      ELSE
             GOTO 500
      ENDIF
 2000 IF(ANS.NE.1) THEN
             GOTO 300
      ENDIF
C
C     ****************************************************************
```

```
C     *AFTER, THE MLSE NEURAL NETWORK HAS ESTIMATED THE N DATA BITS  *
C     *TRANSMITTED, WE ARE NOW GOING TO START CALCULATING THE DIRECT *
C     *MLSE COST FUNCTION ESTIMATES OF THE N TRANSMITTED DATA BITS   *
C     *THEN, WE ARE GOING TO MAKE A COMPARISON BETWEEN THE TWO ESTIMA*
C     *TES AND RECORD THE RESULT.                                    *
C     ***************************************************************
C
  700 DO 180 I = MIN, SUP
            SUM2 = 0
            DO 190 K = MIN, SUP
                  FACTOR = (I - K) * PERIOD
                  SUM2 = SUM2 + A(K) * F(FACTOR,TNOT)
  190       CONTINUE
            YPRIME(I) = SUM2 + GN(I)
  180 CONTINUE
      DO 230 I = MIN, SUP
            SUM3 = 0
            DO 240 K = MIN, SUP
                  FACTOR = (K - I) * PERIOD
                  SUM3 = SUM3 + YPRIME(K) * F(FACTOR,TNOT)
  240       CONTINUE
            ZPRIME(I) = SUM3
  230 CONTINUE
      DO 250 I = MIN, SUP
            SUM4 = 0
            DO 260 K= MIN, SUP
                  FACTOR = (I - K) * PERIOD
                  SUM4 = SUM4 + A(I) * S(FACTOR,TNOT,NNOT) * A(K)
  260       CONTINUE
            MLSECF(I) = 2 * A(I) * ZPRIME(I) - SUM4
  250 CONTINUE
      GOTO 350
  300 J = 1
      DO 470 I = MIN , SUP
             VPRIME(I) = V(J)
             J = J + 1
  470 CONTINUE
      DO 390 I = MIN , SUP
            SUM7 = 0
            DO 410 K = MIN , SUP
                  FACTOR = (I - K) * PERIOD
                  MUL = VPRIME(K)
                  SUM7 = SUM7 + A(K) * FPRIME(FACTOR,TNOT,MUL)
  410       CONTINUE
            YPRIME(I) = SUM7 + GN(I)
  390 CONTINUE
      DO 420 I = MIN , SUP
            SUM8 = 0
            DO 430 K = MIN , SUP
                  FACTOR = (K - I) * PERIOD
                  MUL = VPRIME(K)
                  SUM8 = SUM8 + YPRIME(K) * FPRIME(FACTOR,TNOT,MUL)
  430       CONTINUE
            ZPRIME(I) = SUM8
  420 CONTINUE
      DO 440 I = MIN , SUP
```

```
            SUM9 = 0
            DO 450 K = MIN , SUP
                  FACTOR = (I - K) * PERIOD
                  MUL = VPRIME(K)
                  SUM9 = SUM9 + A(I)*SPRIME(FACTOR,TNOT,NNOT,MUL)*A(K)
  450       CONTINUE
            MLSECF(I) = 2 * A(I) * ZPRIME(I) - SUM9
  440 CONTINUE
  350 IF(SUP.EQ.N) GOTO 400
      MIN = MIN + M
      SUP = SUP + M
      IF(SUP.GT.N) THEN
             SUP = N
      ENDIF
      IF(ANS.EQ.1) THEN
            GOTO 700
      ELSE
            GOTO 300
      ENDIF
  400 DO 280 I = I, N
            IF(AOUT(I).NE.MLSECF(I)) THEN
                   NUMBER = NUMBER + 1
            ENDIF
  280 CONTINUE
      PRINT*,'THE NUMBER OF ERROR DATA BETWEEN THE TRANSMITTED'
      PRINT*,'BINARY SEQUENCE OF DATA AND DATA DIRECTLY GENERATED'
      PRINT*,'BY THE MLSE COST FUNCTION :'
      PRINT*,' '
      PRINT 290,NUMBER
  290 FORMAT(2X,'NUMBER =',4X,I5)
      GOTO 3000
  3   STOP
      END
C
C     ***************************************************************************
C     *HERE IS THE SUBROUTINE CORRESONDING TO THE TRANSMISSION CHANNE*
C     *L IMPULSE RESPONSE WHICH IS MODELED BY A FINITE RESPONSE SQUAR*
C     *ED COSINE FUNCTION. THIS FUNCTION IS IMPLEMENTING THE STATIONA*
C     *RY CHANNEL.                                                   *
C     ***************************************************************************
C
      FUNCTION F(FACTOR,TNOT)
      REAL FACTOR
      REAL F, TNOT, PI, SUP
      PI = 3.1415927
      SUP = TNOT / 2.0
      IF(ABS(FACTOR).LE.SUP) THEN
            F = (COS(PI *FACTOR / TNOT)) ** 2
      ELSE
            F = 0
      ENDIF
      RETURN
      END
C
C     ***************************************************************************
C     *HERE IS THE COMBINED RESPONSE OF THE CHANNEL AND MATCHED      *
```

```
C     *FILTER. THIS FUNCTION IS IMPLEMENTING THE STATIONARY CHANNEL. *
C     ***************************************************************
C
      FUNCTION S(FACTOR,TNOT,NNOT)
      REAL FACTOR, S, TNOT, NNOT, PI
      PI = 3.1415927
      IF(ABS(FACTOR).LE.TNOT) THEN
            S = (1 / (2 * NNOT)) * ((TNOT - ABS(FACTOR))
     &          * (1 + 0.5 * COS(2 * PI * FACTOR / TNOT))
     &          + (3 * TNOT / 4.0) * SIN (2 * PI * ABS(FACTOR) * TNOT))
      ELSE
            S = 0
      ENDIF
      RETURN
      END
C
C     ***************************************************************
C     *HERE ARE THE SET OF DIFFERENTIAL EQUATIONS DESCRIBING THE DYNA*
C     *MICS OF THE NEURAL NETWORK.                                   *
C     ***************************************************************
C
      SUBROUTINE FCN(M,T,U,UPRIME)
      INTEGER M
      REAL U(M), UPRIME(M), T, SOM
      COMMON W(17,17), TAU, CURRENT(17), VIN(17)
      DO 210 I = 1, M
            SOM = 0
            DO 220 K = 1, M
                  SOM = SOM + W(I,K) * VIN(K)
  220       CONTINUE
            UPRIME(I) = SOM - (U(I) / TAU) + CURRENT(I)
  210 CONTINUE
      RETURN
      END
C     ***************************************************************
      SUBROUTINE FCNJ(M,T,U,PD)
      INTEGER M
      REAL U(M), PD(M,M), T
      RETURN
      END
C
C     ***************************************************************
C     *SAME AS FOR THE FUNCTION F ONLY THIS TIME IS FOR THE TIME-VARY*
C     *ING CHANNEL.                                                  *
C     ***************************************************************
C
      FUNCTION FPRIME(FACTOR,TNOT,MUL)
      REAL FACTOR
      REAL FPRIME, TNOT, PI, SUP, MUL
      PI = 3.1415927
      SUP = TNOT / 2.0
      IF(ABS(FACTOR).LE.SUP) THEN
            FPRIME = MUL * ((COS(PI *FACTOR / TNOT)) ** 2)
      ELSE
            FPRIME = 0
      ENDIF
```

```
      RETURN
      END
C
C     ***********************************************************************
C     *SAME AS BEFORE (FOR THE FUNCTION S) ONLY THIS TIME IT IS FOR  *
C     *THE TIME-VARYING CHANNEL.                                     *
C     ***********************************************************************
C
      FUNCTION SPRIME(FACTOR,TNOT,NNOT,MUL)
      REAL FACTOR, SPRIME, TNOT, NNOT, PI, MUL
      PI = 3.1415927
      IF(ABS(FACTOR).LE.TNOT) THEN
             SPRIME = ((MUL ** 2) / (2 * NNOT)) * ((TNOT - ABS(FACTOR))
     &                    * (1 + 0.5 * COS(2 * PI * FACTOR / TNOT))
     &               + (3 * TNOT / 4.0) * SIN (2 * PI * ABS(FACTOR) * TNOT))
      ELSE
             SPRIME = 0
      ENDIF
      RETURN
      END
```

LIST OF REFERENCES

1. DARPA, *Neural Network Study*, AFCEA International Press, November 1988.

2. Lippmann, Richard P., "An Introduction to Computing With Neural Nets," *IEEE, ASSP Magazine*, pp. 4-22, April 1987.

3. Provence, John D., "Neural Network Implementation for Maximum-Likelihood Sequence Estimation of Binary Signals in Gaussian Noise," *1987 IEEE International Conference On Neural Networks*, Vol. 3, pp. 703-714.

4. Lippmann, R. P., Gold, B. and Malpass, M. L., "A Comparison of Hamming and Hopfield Neural Nets for Pattern Classification," Technical Report No. AD-A182 255, MTI, 21 May 1987.

5. Arbib, Michael A., *Brains, Machines and Mathematics*, Springer- Verlag, 1987.

6. Denker, John S., *Neural Networks for Computing*, AIP Conference Proceedings 151, Snowbird, UT, 1986.

7. Grossberg, S., "Adaptive Pattern Classification and Universal Recording: Feedback, Expectation, Olfaction, and Illusions," *Biological Cybernetics*, Vol. 23, pp. 187-202, 1976.

8. Gottfried, Ungerboeck, "Adaptive Maximum-Likelihood Receiver for Carrier-Modulated Data-transmission Systems," *IEEE Transactions on Communications*, Vol. COM-22, pp. 624-636, May 1974.

9. Proakis, John G., *Digital Communications*, pp. 351-352, pp. 394-412, McGraw-Hill, 1983.

10. Hopfield, J. J. and Tank, D. W., "Neural Computation of Decisions in Optimization Problems," *Biological Cybernetics*, Vol. 52, Springer-Verlag, pp. 141-152, 1985.

11. Gottfried, Ungerboeck, "Nonlinear Equalization of Binary Signals in Gaussian Noise," *IEEE Transactions on Communications Theory*, Vol. COM-19, pp. 1128-1137, Dec 1971.

12. Carpenter, Gail A. and Grossberg S., "A Massively Parallel Architecture For a Self-Organizing Neural Pattern Recognition Machine," *Computer Vision, Graphics and Image Processing*, Vol. 37, pp. 54-115, 1987.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Chairman, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Professor Tri T. Ha, Code 62Ha Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	2
5.	Professor R. Janaswamy, Code 62Js Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	2
6.	Etat Major de la Marine Royale Division du Personnel et d'Instruction Rabat / MOROCCO	1
7.	Ecole Royale Navale Librairie Casablanca 01 / MOROCCO	1
8.	Mohamed Hassan Khaidar Rue Souissi, No. 2 Sidi Amar Lahcini Meknes / MOROCCO	4
9.	Shu Shih Ming SMC 2156 Naval Postgraduate School Monterey, CA 93943-5000	1
10.	Al Metlaq Issam SMC 1619 Naval Postgraduate School Monterey, CA 93943-5000	1

Thesis
K39623
c.1

Khaidar
Computer implementation and simulation of some neural networks used in pattern recognition and classification.